Computergestützte Diagnose und Visualisierung am Beispiel von arteriellen Gefäßen in dreidimensionalen CT-Daten

Diplomarbeit

vorgelegt von
Alexander Horn



Institut für Computervisualistik
Arbeitsgruppe Computergrafik

Betreuer: Dipl. -Inform. Matthias Biedermann
Prüfer: Prof. Dr. Stefan Müller

Oktober 2005

Inhaltsverzeichnis

1 Einleitung

In der Medizin gibt es immer mehr bildgebende Verfahren, die digitale Daten erzeugen und verarbeiten. Doch aufgrund des immer größer werdenden Datenaufkommens, ist es für den Arzt immer aufwändiger mit den Daten umzugehen, weshalb eine computergestützte Diagnose (CAD: computer aided/assisted diagnosis) immer mehr an Bedeutung gewinnt. Herkömmliche Bildverarbeitungsverfahren für die Unterstützung der Diagnose arbeiten häufig auf einzelnen, zweidimensionalen Schnittbildern, was im Allgemeinen zu einer guten Performanz dieser Algorithmen führt. Andererseits können räumliche Informationen aus einer Folge solcher Schnittbilder, wie sie beispielsweise bei CT-Untersuchungen anfallen, nur mit erheblichem Aufwand und beschränkter Qualität gewonnen werden. Darüber hinaus spielt bei der diagnostischen Verarbeitung solcher Bilder z.B. die Schnittebene eine große Rolle, so dass eine umfangreiche Verknüpfung verschiedener Arbeitsschritte notwendig wird.

Andererseits ist die Verarbeitung und Visualisierung des kompletten Volumendatensatzes, der aus den gleichen Schnittbildern des Aufnahmeverfahrens einfach erstellt werden kann, durch aktuelle Entwicklungen der (Graphik)-Hardware zunehmend effizienter und flexibler. So können auch komplexere Algorithmen zur Verarbeitung, sowie die aufbereitete Darstellung der gesuchten Informationen direkt auf Basis der Volumendaten durchgeführt werden.

Im Rahmen dieser Diplomarbeit soll am Beispiel von Gefäßerkrankungen untersucht werden, inwieweit eine computergestützte Diagnose auf 3D-Daten sinnvoll und möglich ist. Als Grundlage dienen dabei reale CT-Scans, in denen die Gefäßstrukturen zunächst identifiziert und visualisiert werden sollen. In einem zweiten Schritt sollen – unter Ausnutzung der Besonderheiten des bildgebenden Verfahrens – mögliche pathologische Strukturen gefunden und in geeigneter Weise dargestellt werden.

Hierfür wird ein Programm entwickelt, das CT-Scans einlesen kann und die erwähnten Aufgaben umsetzt. Da es sich bei diesem Programm um ein Segmentierungs- und Betrachtungsprogramm zur computergestützten Diagnose (CAD) handelt, wird das Programm CADView genannt.

2 Grundlagen

Hintergrund dieser Arbeit ist die Tatsache, dass in modernen Gesellschaften Gefäßerkrankungen zu den häufigsten Krankheits- und Todesursachen zählen. Ziel ist es den Arzt bei seiner Diagnose zu unterstützen und mögliche Erkrankungen mit Hilfe der Computergrafik zu erkennen und zu visualisieren. Hierzu wird ein Grundwissen über die Anatomie der Gefäße und deren Erkrankungen, die Computertomografie, Verfahren zur Auffindung von Strukturen in den Bildern und schließlich auch Wissen über Möglichkeiten zur Visualisierung benötigt. Dieses Grundwissen wird im ersten Teil dieser Arbeit vermittelt, um darauf in den weiteren Kapiteln aufbauen zu können.

2.1 Gefäßbaum

Der menschliche Körper wird durch die Arterien mit frischem Blut vom Herzen versorgtWenn das Herz schlägt, also sich zusammenzieht, wird Blut durch die Aorta in die Arterien und somit in den Körper gepumpt. Hierbei weiten sich die Gefäße, damit sie das Blut aufnehmen können. Um einen reibungslosen Blutfluss zu gewährleisten, bestehen die Arterien aus mehreren Schichten. Die innerste Schicht, die Intima, sorgt für den Flüssigkeits- und Stoffaustausch zwischen Blut und dem umliegendem Gewebe. Die Medina ist die mittlere Schicht und besteht aus Muskeln und ist dafür zuständig, dass sich die Arterie der aktuellen Blutmenge im Durchmesser anpassen kann. Die äußerste Schicht ist die Adventitia, welche die inneren Schichten mit einem Bindegewebe umhüllt.

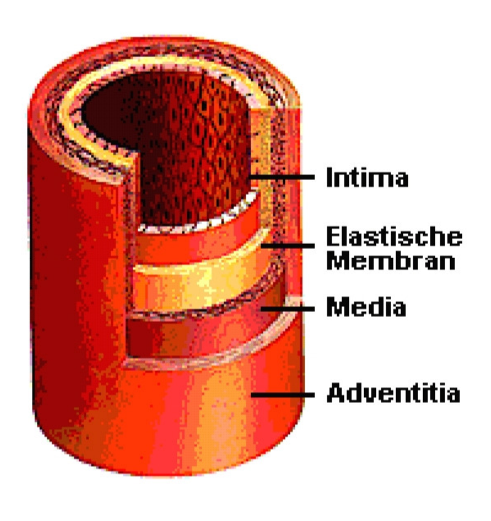


Abbildung 1: Gefäßaufbau

2.1.1 Erkrankungen

Erkrankungen von Gefäßen in Form von Verkalkungen werden auch als Arteriosklerose bezeichnet. Hierbei handelt es sich um eine Verengung der Gefäße, die über einen längeren Zeitraum entstehen und deshalb die Symptome nicht direkt bemerkbar sind. Arteriosklerose beschreibt den natürlicher Alterungsprozess von Gefäßen, der durch Risikofaktoren wie Bluthochdruck oder Rauchen beschleunigt wird. Sie beginnt mit einer Beschädigung der Intima, die bei der normalen Wundheilung aufquillt. Hierdurch entsteht ein Ödem. Durch weitere rote Blutkörperchen und Fresszellen wächst die Verdickung, auch Plaque genannt, weiter an und es lagert sich darin Kalk ab. Je nach dem wie viel Kalk sich ablagert, können die Verdickungen hart oder weich sein. Als Folge der

Plaques verengen sich die Arterien was zu einem kompletten Verschluss des Gefäßes führen kann. Dies bedeutet, dass die Region, die normalerweise durch diese Arterie mit Nährstoffen versorgt wird, keinen Sauerstoff mehr bekommt und absterben kann. Es kommt zu einem Infarkt. Reißen die Plaques hingegen auf, kann es zu einer Narbenbildung kommen, was die Elastizität der Gefäße einschränkt und ebenfalls den Blutfluss behindert.

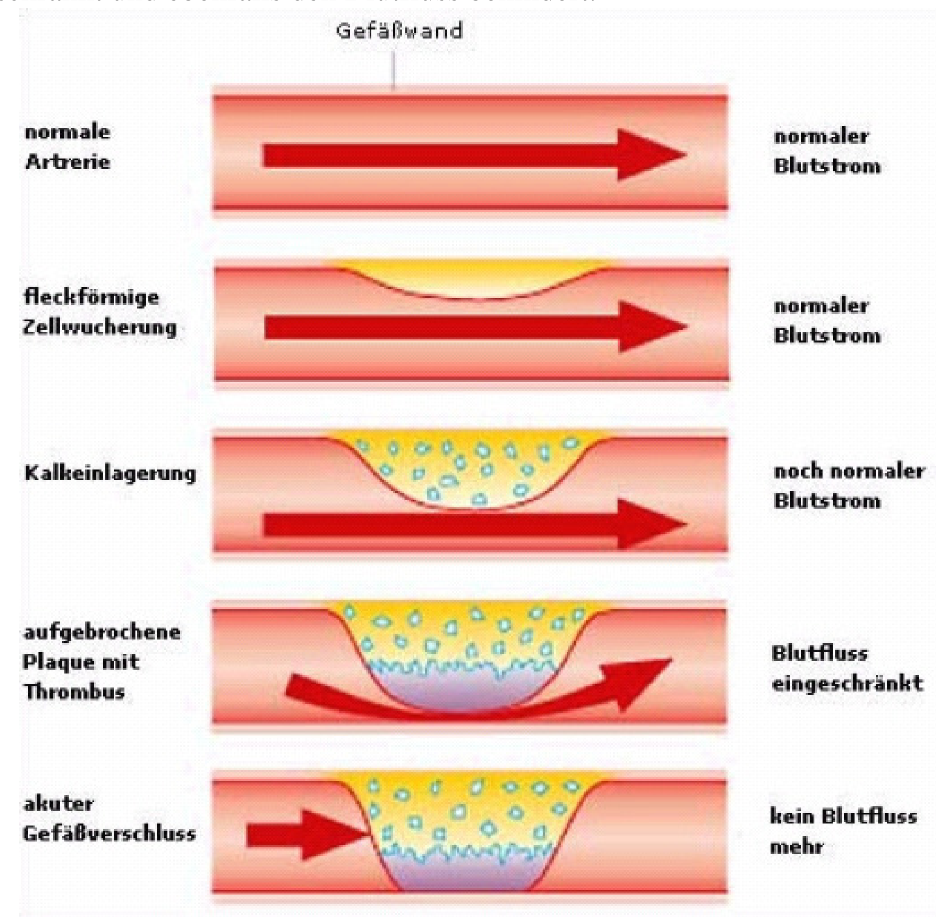


Abbildung 2: Entwicklung eines Gefäßverschlusses

2.2 Computertomographie

Ein bildgebendes Verfahren zur nicht-invasiven [7.2] Diagnose stellt die Computertomographie (CT) dar. Mit ihrer Hilfe können zu untersuchende Körper in Schichten aufgenommen, am Computer zu 3D-Volumendatensätzen zusammengesetzt und schließlich betrachtet werden. Dieses Verfahren ist für die Patienten im Gegensatz zur Herzkatheteruntersuchung ein sehr schonender und schneller Vorgang und bietet einen umfassenderen Blick auf den Gefäßbaum und die Organe.

2.2.1 Funktionsprinzip

Bei der Computertomographie werden Röntgenstrahlen durch den zu untersuchenden Körper geschickt und auf der anderen Seite von mehreren Detektoren wieder aufgefangen. Aus der Differenz von ausgestrahlter zu aufgefangener Strahlung können Rückschlüsse auf das untersuchte Gewebe gezogen werden. Daraus errechnet ein Computer ein Schichtbild. Das nun vorliegende 2D Schichtbild hat abhängig vom Gerätetyp eine Auflösung von 256^2, 512^2 oder 1024^2 Pixel. Je nachdem, welches Ziel man mit der Untersuchung verfolgt, kann der Detailgrad der Aufnahmen durch Verringerung der Abstände der einzelnen Schichtbilder erhöht werden. Meist werden Schichtbilder im Abstand von 1 mm bis 10 mm aufgenommen. Um einen Volumendatensatz zu erhalten ist es nötig viele einzelne Schichten aufzunehmen und danach zu einem Volumen zusammenzusetzen. Um das Verfahren der Aufnahme zu beschleunigen, werden Mehrschicht-Spiral-Computertomographen eingesetzt, die zum einen die Röntgenröhre um den Patienten drehen und gleichzeitig den Patienten schrittweise durch die Röhre hindurch schieben.

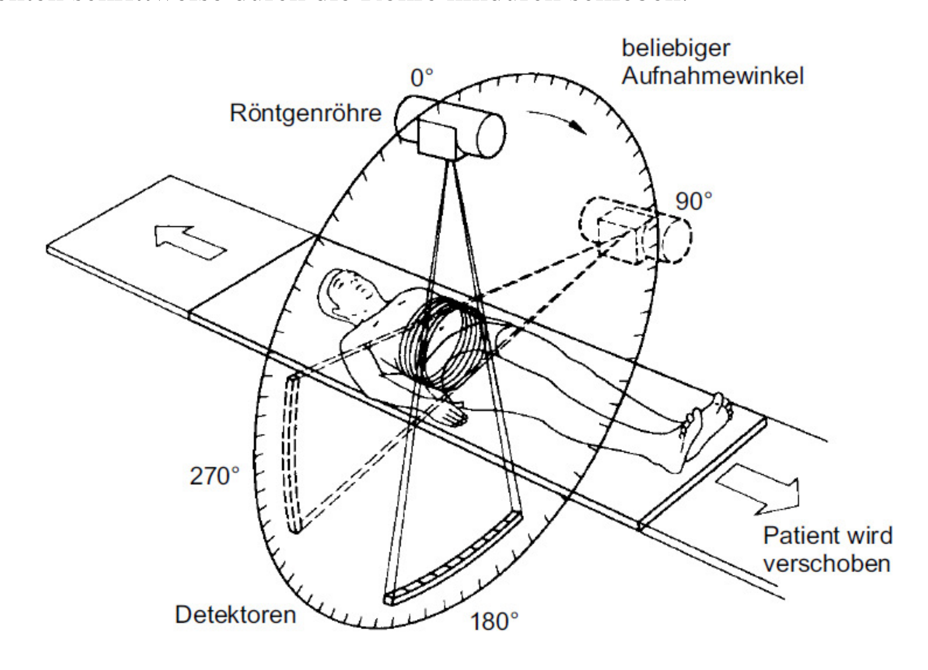


Abbildung 3: Computertomographie

Mit Hilfe dieses Verfahrens ist es möglich 300 und mehr Schichtbilder in nur 9 Sekunden aufzunehmen, während der Patient regungslos und mit angehaltenem Atem durch die Röhre geschoben wird.

Um Bewegungsartefakte durch den Herzschlag zu vermeiden, besteht die Möglichkeit, den Aufnahmezeitpunkt durch den Herzschlag des Patienten zu steuern, so dass immer zur gleichen Phase eine Aufnahme gemacht wird.

2.2.2 Hounsfield-Skala

Die Absorptionswerte des aufgenommenen Gewebes werden in Hounsfield Units (HU) angegeben. Die Einheit wurde nach dem Nobelpreisträger Godfrey Hounsfield[1] benannt, der als Vater der Computertomographie gilt. Es wurde eine Skala angelegt, bei welcher den unterschiedlichen Grauwertstufen unterschiedliche Gewebestrukturen zugeordnet werden können. Als Referenz dient Wasser mit einem Wert von 0 HU, was den Ausgangswert für die Skala bildet.

Durch Angabe eines bestimmten Grauwertes ist es allerdings noch nicht möglich eine bestimmte Struktur zu extrahieren, da die unterschiedlichen Gewebestrukturen teils sehr ähnliche oder gar gleiche Grauwerte aufweisen.

Abbildung 4: Hounsfield Skala

[1] Sir Godfrey Newbold Hounsfield CBE (* 28. August 1919 in Newark in Nottinghamshire; † 12. August 2004 in Kingston upon Thames) war englischer Elektrotechniker.

2.2.3 DICOM-Format

Die Grauwertbilder, die ein Computertomograph liefert, werden üblicherweise in der Medizin in einem speziellen Format gespeichert, das mittlerweile zum Standard für medizinische Aufnahmen geworden ist. Dieses Format nennt sich DICOM („Digital Imaging and Communications in Medicine“). Es findet hauptsächlich Anwendung in der Kernspin- und Computertomographie sowie bei Ultraschall-Aufnahmen.

Das DICOM-Format beschreibt, wie die digitalen Bilder und Informationen gespeichert werden und medizinische Aufnahmegeräte miteinander kommunizieren können. Das Besondere ist hierbei, dass die Bilder einen Header vor dem eigentlichen Bild enthalten, in dem zusätzlich Informationen wie Patientenname, Aufnahmegerät, Bildeigenschaften sowie weitere Informationen dokumentiert werden können und nicht wie bei anderen Formaten, bei denen die Zusatzinformationen in separaten Dateien gespeichert sind.

2.3 Bildverarbeitung

Um aus den im DICOM-Format vorliegenden Schichtbildern Rückschlüsse über anatomische Strukturen und Erkrankungen ziehen zu können, gibt es zwei Möglichkeit: Entweder man betrachtet jedes Schichtbild einzeln, oder man bedient sich der Computergrafik und Bildverarbeitung, um die Datensätze als Volumen zu betrachten und damit zu arbeiten zu können. Die Arbeit mit den einzelnen Schichtbildern ist zwar im Gegensatz zur Arbeit mit Volumendatensätzen der Computergrafik weniger fehleranfällig, doch besteht dabei für den Betrachter die Schwierigkeit, sich die dreidimensionalen Strukturen in den zweidimensionalen Schichtbildern vorzustellen. So kann es für den Betrachter recht kompliziert werden den Verlauf einer bestimmten Struktur in mehreren Schichtbildern zu verfolgen und daraus Rückschlüsse zu ziehen.
Die Computergrafik löst dieses Problem, indem sie Strukturen dreidimensional darstellt und der Benutzer damit frei interagieren kann. Man differenziert zwischen zwei unterschiedlichen Methoden der Visualisierung. Zum einen bietet sich die Möglichkeit, ähnlich wie bei den einzelnen Schichtbildern, mit Schnittebenen im Volumenbild zu arbeiten. Hierbei behält man mittels der Schnittebenen durch alle Raumachsen immer noch einen Bezug zum kompletten Volumen. Zum anderen besteht die Möglichkeit, sich spezielle Strukturen separat anzeigen zu lassen. Diese müssen hierzu allerdings aus dem Volumen herausgefiltert werden. Die Bildverarbeitung bietet mehrere Möglichkeiten eingelesene Volumendatensätze aufzubereiten und somit für die folgenden Schritte zu optimieren und aus den aufbereiteten Volumen Strukturen zu segmentieren. Diese werden später gesondert zu bearbeitet oder visualisiert.

2.3.1 Aktueller Bezug

Einen aktuellen Ansatz dazu bietet das Paper von Anthony Sherbondy „Fast Volume Segmentation With Simultaneous Visualization Using Programmable Graphics Hardware“ [2], in dem versucht wird Strukturen aus Volumen zu segmentieren und die Visualisierung der Segmente durch moderne programmierbare Grafikhardware zu beschleunigen, indem die Berechnungen auf der Grafikhardware stattfinden. Durch dieses Vorgehen soll es dem Benutzer ermöglicht werden in Echtzeit das Ergebnis seiner Segmentierung zu begutachten und gegebenenfalls Parameter

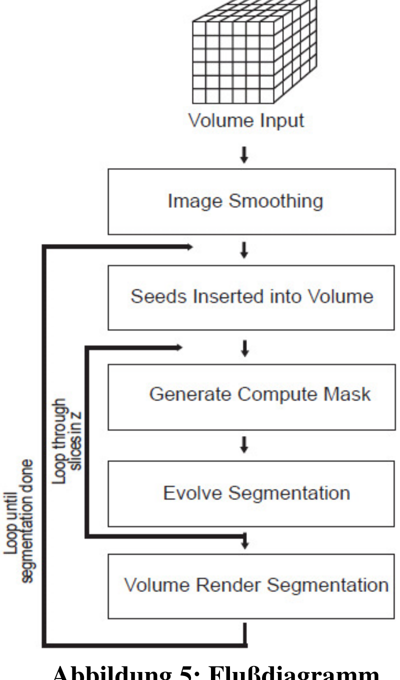


Abbildung 5: Flußdiagramm Sherbondy

direkt zu ändern, sollte das Ergebnis nicht seinen Wünschen entsprechen.
Das Verfahren beruht auf mehreren Schritten: Zuerst wird das Volumen bei der Vorverarbeitung mittels eines nichtlinearen Filter von Rauschen befreit. Nächster Schritt ist die Segmentierung mit Hilfe einer Region-Growing-Segmentierung [7.2], welche auf Grauwerten und Gradienten basiert. Abschließend erfolgt die Visualisierung der Daten.

In dieser Arbeit, wird der Schwerpunkt auf verschiedene Möglichkeiten der Extrahierung der Blutbahn und deren Verkalkungen gelegt und nicht wie bei Sherbondy auf eine möglichst schnelle Visualisierung der Strukturen. Aus diesem Grund ist es nicht unbedingt notwendig die Berechnungen wie bei Sherbondy auf der GPU auszuführen, sondern wie üblich auf der CPU. Ähnlichkeiten zwischen den verschiedenen Zielsetzungen wird es jedoch bei den ersten beiden Schritten der Bildverarbeitung geben: Bei der Vorverarbeitung mit kantenerhaltenden Filtern sowie der Segmentierung mit dem Region-Growing-Algorithmus.

2.3.2 Vorverarbeitung

Digital aufgenommene Bilder besitzen ein gewisses Maß an Unschärfe, welche auf der Tatsache beruht, dass die Bildinformationen beim Aufnahmeverfahren auf Pixel aufgeteilt werden müssen und dass bei der Aufnahme Störsignale entstehen. Diese Unschärfe wird als Bildrauschen bezeichnet. Um mit den vom Computertomographen erzeugten Bildern korrekt arbeiten zu können, ist es deshalb notwendig, die einzelnen Bilder vor ihrer Weiterverarbeitung vorzubereiten. Ziel ist es hierbei das Rauschen aus den Bildern zu entfernen und die Bilder zu glätten, um später eine bessere Segmentierung durch homogene Flächen zu erreichen. Hierzu gibt es unterschiedliche Filter die verschiedenen Ziele verfolgen.



Abbildung 6: Bildrauschen

2.3.2.1 Lineare Filter

Lineare Filter, wie zum Beispiel der Mittelwertfilter oder der Wiener Filter, charakterisieren sich dadurch, dass sie den neuen Farbwert eines Pixels berechnen, indem sie eine Gewichtungsmaske mit der darunter liegenden Region multiplizieren. Die Gewichtungsmaske ist dabei symmetrisch und kann durch eine Faltung dargestellt werden. Mit linearen Filtern kann man Bilder komplett glätten. Allerdings haben sie dadurch auch den Nachteil, dass Kanten

mitgeglättet werden, die für spätere Schritte, wie z.B. für die Segmentierung, benötigt werden

2.3.2.2 Nichtlineare Filter

Nichtlineare Filter hingegen sind nicht durch eine Faltung zu beschreiben und können, abhängig vom Aufbau des Filters, die Bilder glätten ohne dass eine Kantenglättung erfolgt. Zu dieser Art von Filtern zählt zum Beispiel der Medianfilter.

2.3.2.3 Adaptive Filter

Die adaptiven Filter bewirken eine Rauschunterdrückung ohne dabei andere wichtige Strukturen wie Kanten zu zerstören. Der Filter passt sich der Umgebung an, die er filtern soll und kann dadurch besser auf die Umgebung eingehen und relevante Objekte erhalten.

2.3.3 Segmentierung

Sind die Bilder durch die Filter geglättet, so kann man mit der eigentlichen Bearbeitung beginnen. Hierbei ist die Segmentierung ein wichtiger Schritt. Bei der Segmentierung werden benachbarte Pixel (2D) oder Voxel (3D) auf Homogenitätskriterien untersucht. Bei Gemeinsamkeiten wird versucht zusammenhängende Objekte zu bilden. Hierbei dienen die Farbwerte als Kriterium zur automatischen Segmentierung. So werden Schwellwerte angegeben und die Pixel beziehungsweise Voxel bezüglich der Schwellwerte getestet. Hierzu gibt es verschiedene Methoden, die unterschiedliche Wege gehen und dementsprechend auch unterschiedliche Ergebnisse liefern.

2.3.3.1 Grundlegende Verfahren

- **Pixelorientierte Verfahren**:

  Es wird der Farbwert jedes Voxels bezüglich der Schwellwerte getestet. Liegt der Farbwert innerhalb des Schwellwertbereiches wird das Voxel zum Segment hinzugefügt. Andernfalls nicht. Es werden allerdings keine zusammenhängenden Segmente zurückgeliefert, da nur jedes Voxel einzeln und nicht in Beziehung zu seinen Nachbarn betrachtet wird.

- **Kantenorientierte Verfahren**:

  Ausgehend davon, dass eine zu segmentierende Struktur einen anderen Grauwert als seine benachbarte Struktur aufweist, bedeutet die Kante zwischen den benachbarten Strukturen einen Grauwertsprung. Ziel ist es eine geschlossene Kante zu finden, die ein Objekt umgibt. Dazu werden im Bild Kanten und Übergänge gesucht und diese zu geschlossenen Formen

zusammengefügt. Voraussetzung hierfür ist natürlich eine gut erkennbare Kante, mit der die Segmentierung gestartet werden kann. Sind die Kanten nur schlecht zu erkennen, da sie verrauscht oder nicht geschlossen sind, kann es zu Fehlern bei der Segmentierung kommen.

- **Regionenorientierte Verfahren**:

  Hierbei wird versucht benachbarte Bildpunkte einer zusammenhängenden Region hinzuzufügen, wenn sie den gegebenen Kriterien entsprechen. Ein weit verbreiteter Ansatz ist hierbei das Region-Growing-Verfahren. Hierbei wird ein Pixel beziehungsweise Voxel als Startpunkt ([7.2] Seedpoint) angegeben und von dort aus die Nachbarpixel auf Ähnlichkeiten hin untersucht. Weichen diese nicht weiter als der angegebene Schwellwert vom Wert des Startpunktes ab, werden sie der Region zugeteilt.

- **Texturorientierte Verfahren**:

  Bei Strukturen, die keine einheitlichen Grauwerte aufweisen, nützen die zuvor genannten Segmentierungsverfahren nichts, da sie auf der Ähnlichkeit der Grauwerte basieren. Hier bietet sich nur die Möglichkeit, die Strukturen mit vorhandenen Texturen zu vergleichen und dann gegebenenfalls die Struktur aufgrund ihrer Ähnlichkeit zur Textur dem Segment hinzuzufügen.

2.4 Software

Nachfolgend werden die verwendeten Softwaresysteme genannt, sowie ihre Merkmale kurz umschrieben, die zur Erstellung der Anwendung verwendet wurden.

2.4.1 Insight Segmentation and Registration Toolkit (ITK)

ITK ist ein open-source Toolkit zur Segmentierung und Registrierung von digitalen medizinischen Datensätzen, die vornehmlich aus CT- oder MR-Scannern gewonnen werden.
Es verwendet eine Pipelinestruktur und setzt auf vorhandenen Klassen auf, in denen eine Vielzahl von Filtern vordefiniert sind. Die Filter sind einfach anzuwenden und nur noch mit den entsprechenden Parametern zu füllen sind.
So stehen beispielsweise Filter zum Einlesen und Schreiben von Datensätzen, verschiedenste Glättungsfilter, mehrere Segmentierungsalgorithmen und andere zur Verfügung. ITK bietet allerdings nicht die Möglichkeit der Visualisierung. Hierfür wird das Visualization Toolkit verwendet.

2.4.2 Visualization Toolkit (VTK)

VTK ist ein frei verfügbares Softwaresystem für 3D Computergrafik, Bildverarbeitung und Visualisierung. Es wird häufig in der Forschung und Entwicklung eingesetzt, um mit Hilfe der C++-Klassenbibliotheken komplexe 2D- beziehungsweise 3D- Grafiken zu berechnen und darzustellen.
Um plattformunabhängig zu bleiben, verwendet VTK zur Darstellung der Grafiken OpenGL. Neben der Möglichkeit in VTK mit C++ zu programmieren, besteht auch noch die Möglichkeit mit Java, TCL und Python zu arbeiten.
VTK kann grob in zwei Einheiten unterteilt werden. Zum einen gibt es das *graphics model*, das die Umsetzung der Computergrafik-Pipeline darstellt und wie GLUT-Funktionen das Grafikfenster zur Verfügung stellt. Zum anderen besteht es aus der Visualisierungspipeline. Diese setzt sich aus einzelnen Datenobjekten zusammen, die sich aus virtuellen Klassen ableiten lassen.

2.4.3 Fast Light ToolKit (FLTK)

FLTK ist ein plattformunabhängiges Toolkit zur Erstellung von Benutzeroberflächen. Es bietet moderne Funktionalitäten ohne die erzeugten Programme unnötig zu belasten. Dabei ist es recht kompakt gehalten und arbeitet mit *shared libraries*. FLTK bietet auch die Möglichkeit mit einem Benutzeroberflächendesigner (FLUID) Oberflächen innerhalb kürzester Zeit zu erstellen. Das Toolkit wird unter der GNU *Library General Public License* angeboten und ist somit frei verfügbar.

2.4.4 CMake

CMake wird benötigt, um die drei zuvor genannten plattform-unabhängigen Toolkits für die eigene Plattform zu erstellen. Es arbeitet mit compiler-unabhängigen Konfigurationsdateien und erzeugt daraus Projektdateien für den Compiler

2.4.5 MS Visual Studio .NET

Als Entwicklungsumgebung wurde Microsofts Visual Studio .Net 2003 gewählt. Es enthält Module zur Entwicklung von C++ und kann die von CMake generierten Projektdateien einlesen, wodurch schon alle Verknüpfungen für das Projekt zu ITK, VTK und FLTK gemacht werden.

3 Softwaretechnischer Entwurf

Die Toolkits VTK und ITK arbeiten nach dem Pipelineprinzip. Sie übergeben also das Ergebnis des einen Arbeitsschrittes an den nächsten weiter. Auf dem Pipelineprinzip aufbauend wird nun ein softwaretechnischer Entwurf eines Ablaufdiagramms erstellt, um zu zeigen wie das zu entwickelnde Programm funktionieren soll und wie die Daten darin fließen.

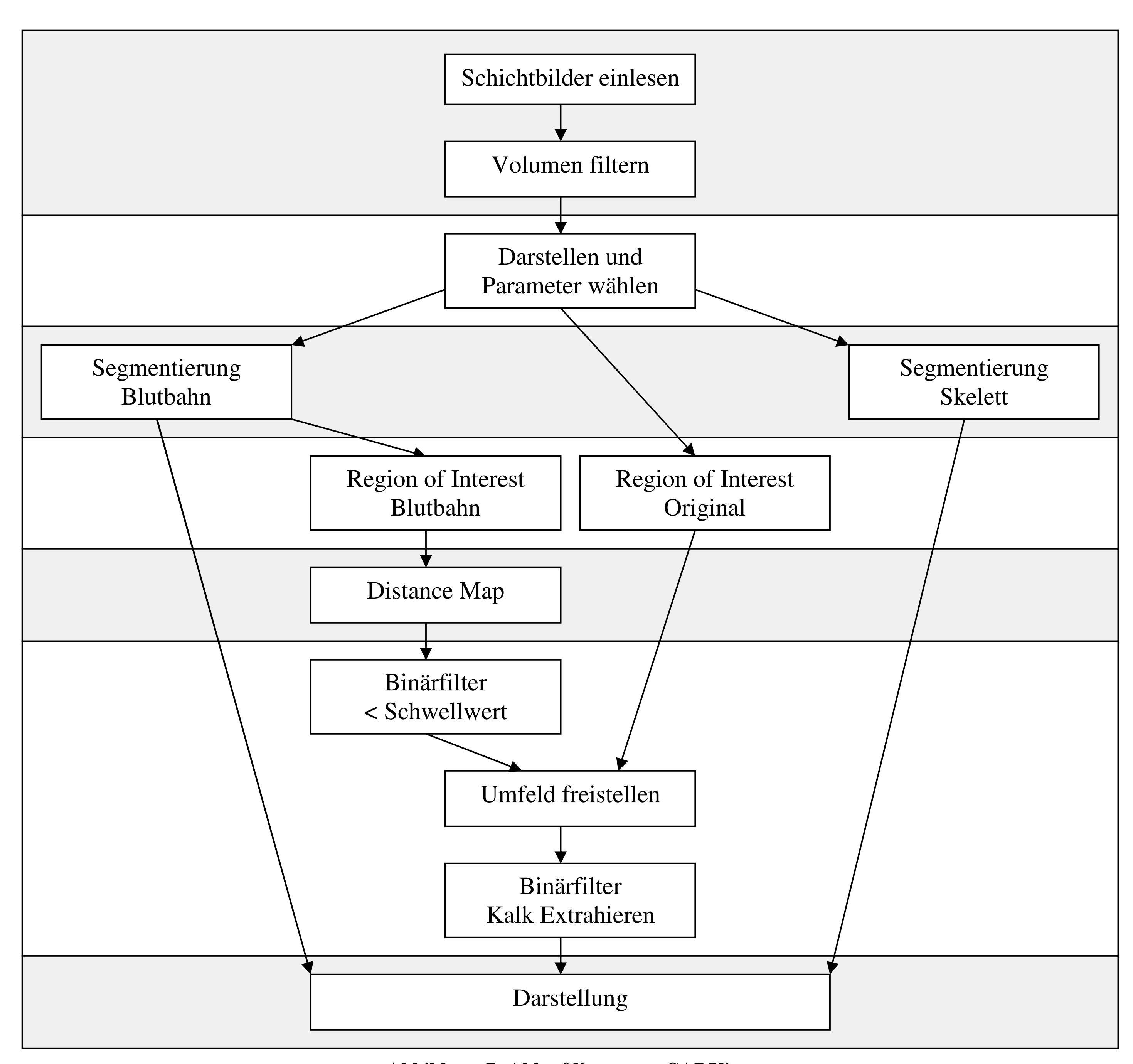


Abbildung 7: Ablaufdiagramm CADView

Der Ablauf kann grob in sieben Einheiten unterteilt werden, in denen ähnliche oder zusammenhängende Funktionen ausgeführt werden. Die einzelnen Schritte sind im Diagramm farblich voneinander abgesetzt, um sie leichter unterscheiden zu können.

- Der erste Schritt besteht darin die Schichtbilder einzulesen und mit einem Glättungsfilter zu bearbeiten.
- Im nächsten Schritt soll das gerade eingelesene Volumen dargestellt werden, damit man Parameter für die darauf folgenden Schritte wählen kann.
- Als drittes werden aufgrund der gewählten Parameter die Blutbahn und das Skelett segmentiert.
- Aus der Blutbahn und dem originalen Volumen werden im nächsten Schritt die *Regions of Interest* [7.2] ausgeschnitten, um den Aufwand geringer zu halten.
- Im fünften Schritt wird dann mit Hilfe der *Region of Interest* eine *DistanceMap* [7.2] der Blutbahn erstellt.
- Darauf folgend wird im sechsten Schritt eine Region um die Blutbahn mit einem bestimmten Abstand binarisiert [7.2]. Diese Region wird dann mit der originalen *Region of Interest* multipliziert, um nur das Umfeld um die Blutbahn herum zu erhalten. Darin wird dann wiederum im Grauwertbereich von Kalk binarisiert, um die Verkalkungen zu bekommen.
- Abschließend kann man sich im letzten Schritt die Blutbahn, die Verkalkungen und das Skelett darstellen lassen.

Im Folgenden werden die einzelnen Schritte mit detaillierteren Diagrammen näher beschrieben und erklärt, was sie als Eingabe benötigen und als Ausgabe zurückgeben.
Allen Schritten, bis auf die Schritte, in denen nur dargestellt wird, ist gemein, dass zu Beginn ein oder mehrere Volumen eingelesen werden und zum Schluss die neu erzeugten Volumen gespeichert werden. Hierzu müssen bei jedem Schritt die Dateinamen der zu ladenden oder zu speichernden Volumen angegeben werden.

3.1 Datensatz einlesen

- **Schichtbilder einlesen**

  Beim Einlesen und abschließendem Speichern eines Satzes von Schichtbildern im DICOM-Format ist es am sinnvollsten alle Schichtbilder mit einem fortlaufenden Index im Dateinamen zu versehen und in einen Ordner zu kopieren in dem sich außer den Bildern nichts anderes befindet.

Dem Reader kann man dann den Speicherort der Bilder angeben, von wo aus die Bilder geladen werden sollen.
Um nicht von einem Format abhängig zu sein, wäre es ratsam, ein zweites Dateiformat zu unterstützen. DICOM gilt zwar als Standard, doch finden sich häufig auch Datensätze in anderen Formaten wie beispielsweise RAW.

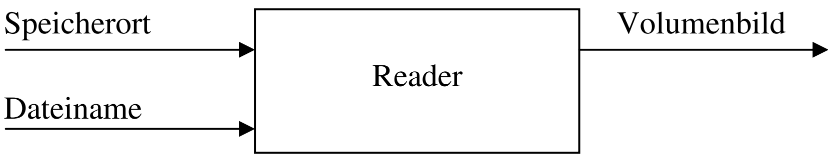


Abbildung 8: Schichtbilder einlesen

Der Reader liest die Schichtbilder ein und macht daraus ein Volumenbild, welches er unter einem angegebenen Namen in einem Format, das die Anforderungen an die folgenden Schritte unterstützt, wieder abspeichert.

- **Volumen filtern**

Nach dem Einlesen sollte das Volumenbild an einen Glättungsfilter übergeben werden. Ziel ist es hierbei, wie schon in den Grundlagen 2.3.2 beschrieben wurde, das Bild von Rauschen zu befreien und somit gute Voraussetzungen für die folgenden Schritte wie die Segmentierung zu schaffen.

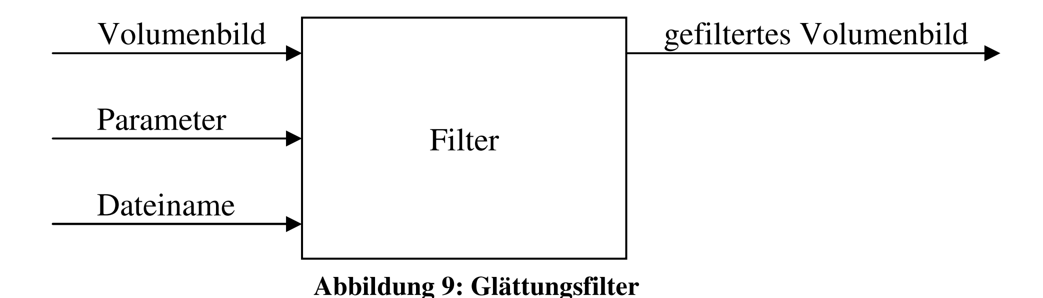


Abbildung 9: Glättungsfilter

Dem Filter muss das eingelesene Volumenbild und verschiedene Filterparameter, abhängig vom Filter, übergeben werden. Diese geben an, wie der Filter seine Arbeit verrichten soll.

3.2 Darstellung und Parameter wählen

Als nächstes wird eine Darstellung benötigt, in der die Parameter für die folgenden Schritte gewählt werden können. Hierzu wird als Eingabe nur der Dateiname des darzustellenden Volumens benötigt.

Es soll in einer Drei-Ebenendarstellung visualisiert werden, um auf jede einzelne Schicht Zugriff zu haben und somit die passenden Parameter für die weiteren Schritte exakt bestimmen zu können.

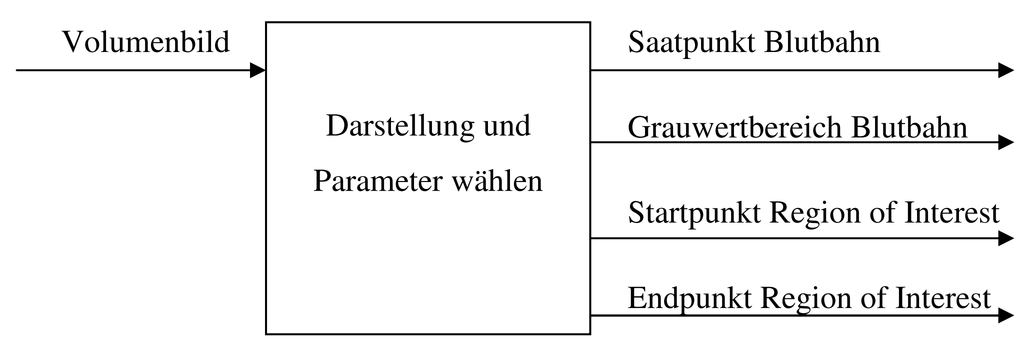


Abbildung 10: Darstellung und Parameter

Als Parameter werden für die nächsten Schritte der Saatpunkt [7.2] und der Grauwertbereich der Blutbahn und der Start- und Endpunkt der *Region of Interest* benötigt.

3.3 Segmentierung der Blutbahn und Skelett

Die Segmentierung der Blutbahn ist neben der Extrahierung der Verkalkungen der wichtigste Schritt, da schließlich die Verkalkungen der Blutbahn untersucht werden sollen. Deswegen ist hier eine genaue Segmentierung notwendig. Für diese Segmentierung eignet sich am besten ein Algorithmus, der regionenorientiert arbeitet, wie es in den Grundlagen 2.3.3.1 beschrieben wurde.
Hierzu muss das Volumenbild weitergegeben werden. Neben weiteren Parametern benötigt der Algorithmus vor allem einen Saatpunkt, von dem aus die Segmentierung starten soll. Je nachdem, welches regionenorientierte Segmentierungsverfahren zum Einsatz kommt, werden weitere Parameter wie Schwellwerte oder Ähnlichkeitsabweichungen benötigt.

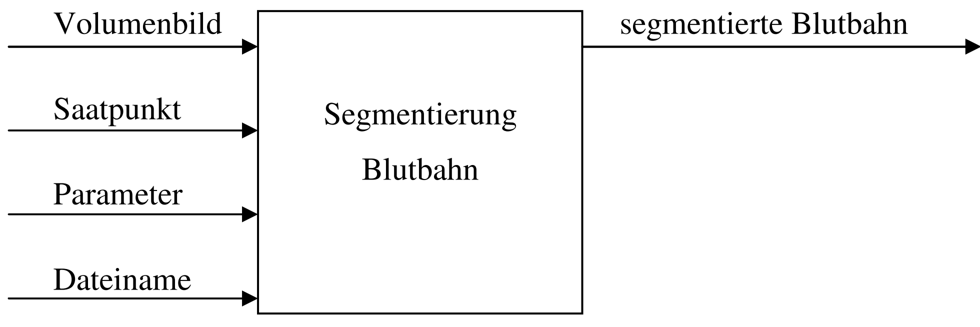


Abbildung 11: Segmentierung Blutbahn

Zur Segmentierung des Skelettes wird ebenfalls das geglättete Volumenbild benötigt. Da das Skelett allerdings keine wichtige Rolle spielt, ist eine richtige Segmentierung hier nicht nötig. Es genügt die Knochen aufgrund der zu Knochen passenden Grauwerte zu segmentieren. Somit werden einfach nur Minimum/Maximum Schwellwerte angegeben, um das Skelett zu segmentieren.

Hierfür genügt ein pixelorientiertes Verfahren zur Segmentierung, wie es ebenfalls in den Grundlagen 2.3.3.1 beschrieben wurde.

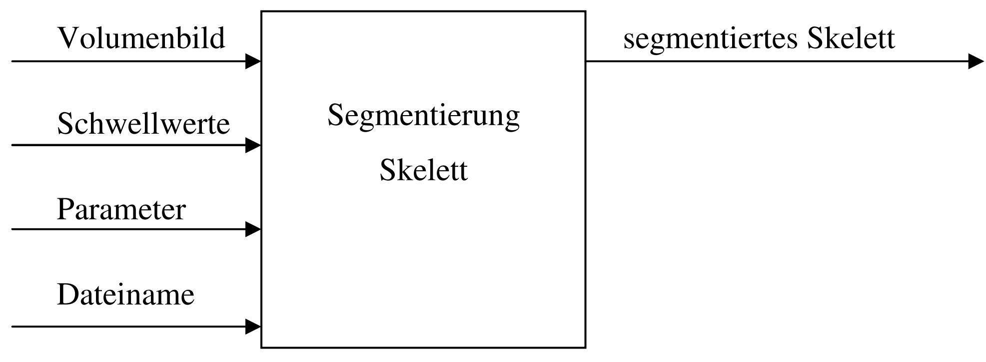


Abbildung 12: Segmentierung Skelett

3.4 Region of Interest erstellen

Um den Rechenaufwand der folgenden Schritte zu minimieren sollte mit *Regions of Interest* [7.2] weitergearbeitet werden. In den folgenden Schritten werden das originale Volumen und das Blutbahnvolumen benötigt. Deshalb sollten diese auf die *Region of Interest* verkleinert werden. Hierzu werden die beiden Volumen und der Startpunkt sowie die Ausdehnung der Region benötigt, die man zuvor bei der Darstellung ermittelt hat.

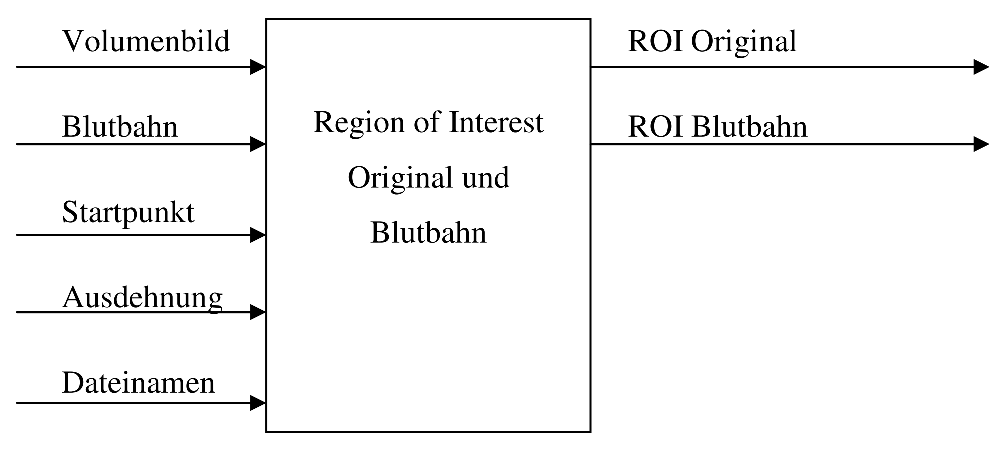


Abbildung 13: Region of Interest

Als Ausgabe erhält man die beiden Volumen der Regionen, die unter den angegebenen Dateinamen gespeichert werden.

3.5 *DistanceMap* erstellen

Da eine gezielte Segmentierung der Verkalkungen nicht effizient wäre, da diese teilweise recht klein und über die gesamte Blutbahn verstreut sind, werden sie mittels eines Schwellwertverfahrens (2.3.3.1) extrahiert. Würde man diese Methode auf das komplette Volumen anwenden, würde man neben den Verkalkungen auch noch das Skelett erhalten, da dieses im gleichen Grauwertbereich liegt. Aus diesem Grund muss die Region, in der nach

Verkalkungen gesucht wird, auf das Umfeld der Blutbahn eingeschränkt werden. Die *DistanceMap* wird benötigt, um eine Region in einem bestimmten Abstand zur Blutbahn festlegen zu können. Es wird dazu das Binärbild der *Region of Interest* der Blutbahn benötigt.

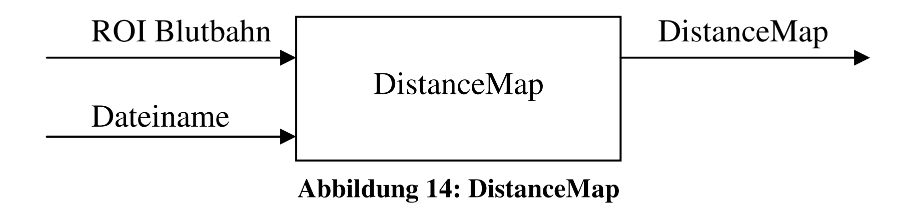


Abbildung 14: DistanceMap

Als Ausgabe erhält man die *DistanceMap*, die im nächsten Schritt weiterverarbeitet wird.

3.6 Verkalkung extrahieren

Durch die *DistanceMap* erhält man den Abstand jedes Voxels zum nächst gelegenen Voxel der Blutbahn und kann somit durch einen Binärfilter nur die Voxel auswählen, die in einem bestimmten Abstand zur Blutbahn liegen. Hierzu wird die *DistanceMap*, der Abstand als Schwellwert und ein Farbwert benötigt, auf den die entsprechenden Regionen gesetzt werden sollen.

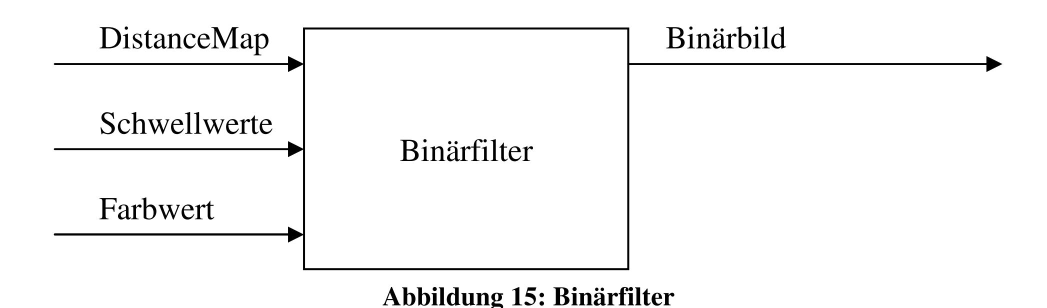


Abbildung 15: Binärfilter

Als Ausgabe erhält man ein Binärbild, in dem alle Voxel innerhalb des angegebenen Abstandes auf den Farbwert gesetzt wurden und alle Voxel außerhalb des Abstandes den Farbwert „0“ besitzen.

Als nächstes benötigt man die original Farbwerte aus dem gerade binarisierten Bereich. Hierzu wird die *Region of Interest* des originalen Volumens mit dem Binärbild multipliziert. Man benötigt das Binärbild und die *Region of Interest* des originalen Volumens.

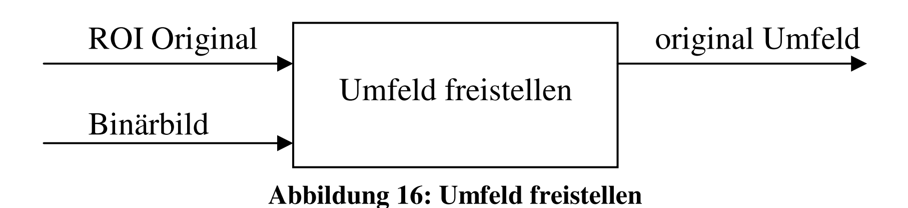


Abbildung 16: Umfeld freistellen

Hat man im letzten Schritt die originalen Farbwerte des Umfeldes der Blutbahn freigestellt, können darin mit einem Binärfilter alle Grauwerte, die Verkalkungen entsprechen, extrahiert werden. Hierzu benötigt man das Umfeldvolumen, Schwellwerte der Verkalkungen und die Farbwerte, auf die diese gesetzt werden sollen.

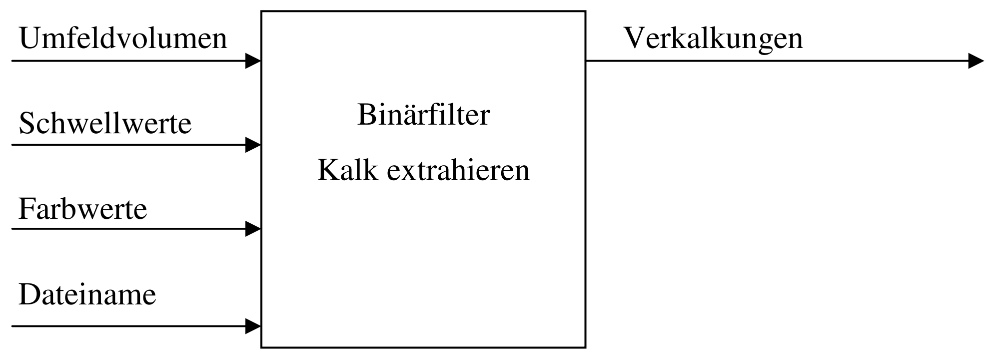


Abbildung 17: Binärfilter Kalk Extrahieren

3.7 Darstellung

Aus dem Schritt Segmentierung hat man nun die Blutbahn und das Skelett erhalten und aus dem Schritt Verkalkung extrahieren die Verkalkungen. Es liegen nun alle zur computergestützten Diagnose der Verkalkungen benötigten Volumen vor, so dass diese visualisiert werden können. Hierzu werden die drei Volumen an die Visualisierung übergeben und dargestellt.

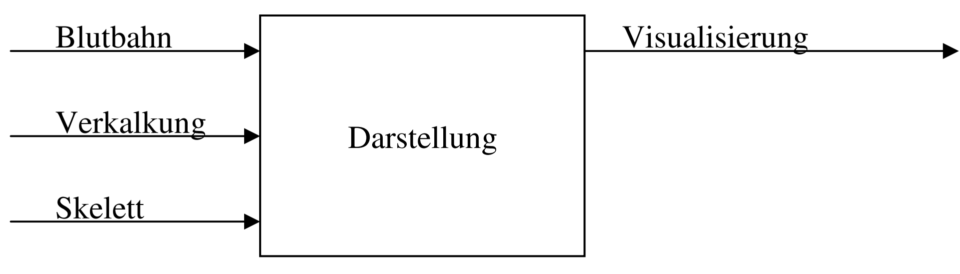


Abbildung 18: Darstellung

Als Ergebnis bekommt man eine Visualisierung der drei Volumen. Sie sollte frei im Raum drehbar und skalierbar sein, um alle Strukturen von allen Seiten und bis ins Detail begutachten zu können.

4 Umsetzung und Visualisierung

Im Folgenden wird nun genauer auf die Umsetzung der einzelnen Funktionen mittels der benutzten Umgebungen ITK und VTK eingegangen. Es werden dabei die verschiedenen Filter vorgestellt, welche in ITK zur Lösung der unterschiedlichen Probleme zur Verfügung stehen, und ihre Parameter und Arbeitsweisen näher erläutert.

4.1 Allgemeine Funktionen

Zu Beginn werden die verwendeten Datentypen und ein paar allgemeine Funktionen erläutert, die in mehreren anderen Funktionen Verwendung finden. Sie werden hier ausführlich beschrieben. Später, bei ihrer Verwendung in den anderen Funktionen, wird nur erwähnt, welche Parameter sich geändert haben.

4.1.1 Datentypen

Datentypen die Verwendung finden sind `float`, `signed short` und `unsigned char`. Hieraus werden die unterschiedlichen `PixelTypen` definiert.

```
typedef float           InputPixelType;
typedef signed short    InternalPixelType;
typedef unsigned char   SegmentedPixelType;
```

Je nachdem, welche Informationen gespeichert werden, kann der Typ mit dem geringsten Speichervolumen gewählt werden. Beim Einlesen und Filtern der Datensätze benötigt man noch den Datentyp `float`, mit dem die Berechnungen des Glättungsfilters möglich sind. Zum Speichern der Volumen genügt es wenn `signed short` verwendet wird, um Speicherplatz zu sparen. Für RAW-Datensätze würde die vorzeichenlose Variante von `short` genügen, doch bei DICOM-Daten wird die vorzeichenbehaftete Variante `signed short` benötigt, da hier auch Grauwerte im negativen Bereich möglich sind. Die Farbwerte können hierbei zwischen -32.767 und 32.767 liegen. Für segmentierte Bildinformationen, wie die Blutbahn oder die Verkalkungen, genügt der Typ `unsigned char` mit Werten zwischen 0 und 255, da hier nur binäre Werte gespeichert werden.

Aus diesen zuvor definierten `PixelTypen` kann man nun die `ImageTypen` definieren, die zusätzlich zur Information über die `PixelTypen` auch noch Information über die Dimension der Bilder enthalten.

```
typedef itk::Image<     InputPixelType,
                        3>
                        InputImageType;

typedef itk::Image<     InternalPixelType,
```

```
                              3>
                              InternalImageType;

    typedef itk::Image<       SegmentedPixelType,
                              3 >
                              SegmentedImageType;

    typedef itk::RawImageIO<InputPixelType,
                              3>
                              RawImageIOType;
```

4.1.2 Reader und Writer

Generell ist allen Schritten gemeinsam, dass zu Beginn ein oder mehrere Datensätze eingelesen werden und nach der Bearbeitung durch die Funktion wieder unter einem anderen Dateinamen abgespeichert werden. Dies hat zwar den Nachteil, dass das Programm einen etwas größeren Rechenaufwand durch das Einlesen und Speichern hat, man aber mehr Kontrolle über die einzelnen Zwischenergebnisse bekommt. Dieser Mehraufwand kann durch gezielte Wahl der Datentypen der zu speichernden Datensätze und Verkleinerung der Volumen durch Reduzierung auf *Regions of Interest* minimiert werden.

Als Dateiformat für die Zwischenschritte, in denen die Datensätze gespeichert und wieder eingelesen werden, wurde das Format *.vtk gewählt, da es für 3D Datensätze geeignet ist und sowohl von ITK als auch VTK unterstützt wird.

Wie der `reader` und der `writer` generell funktionieren, wird am Beispiel des `InputReaderType` und des `InternalWriterType` erklärt und ist dann auf die anderen Datentypen übertragbar. Sie verwenden alle die Klassen `itkImageFileReader` beziehungsweise `itkImageFileWriter` zum Lesen beziehungsweise Schreiben der Dateien.

4.1.2.1 Reader

ITK stellt zum Einlesen eines Datensatzes den `itkImageFileReader` zur Verfügung, der für viele gängige Dateiformate anpassbar ist. Um diesen verwenden zu können muss als erstes die entsprechende Headerdatei in das Programm inkludiert werden.

```
#include "itkImageFileReader.h"
```

Nachdem der `itkImageFileReader` nun bekannt ist, kann man sich einen `PixelType`, der Angaben über den Datentyp enthält, einen ImageType, der den `PixelType` und Informationen über die Dimension enthält, und schließlich den `ReaderType`, der dem `itkImageFileReader` den `ImageType` zuweist, definieren.

```
typedef signed short                         InternalPixelType;
typedef itk::Image<InternalPixelType,3>      InternalImageType;
```

```
typedef itk::ImageFileReader<InternalImageType>
   InternalReaderType;
```

Der eigentliche `reader` besteht nur aus der Erstellung einer neuen Instanz des InputReaderType, der Übergabe des Dateinamens durch `SetFileName()` und der `Update()` Funktion, die ihn zur Ausführung veranlasst.

```
InternalReaderType::Pointer reader =
   InternalReaderType::New();
reader -> SetFileName(Filename);
reader -> Update();
```

Danach ist das eingelesene Volumen verfügbar und kann von darauf folgenden Funktionen mittels `reader -> GetOutput()` ausgelesen werden.

4.1.2.2 Writer

Zum Speichern eines in ITK bearbeiteten Datensatzes bietet ITK den `itkImageFileWriter` an, mit dem in einer Vielzahl von Dateiformaten gespeichert werden kann. Der `itkImageFileWriter` ist genauso wie der `itkImageFileReader` aufgebaut. Man muss ebenfalls als erstes die Headerdatei inkludieren.

```
#include "itkImageFileWriter.h"
```

Bei der Definition des `InternalWriterType` wird ebenfalls wieder ein `ImageType` zugewiesen. In diesem Fall ist es der `InternalReaderType`, der nicht nur zum Einlesen sondern auch zum Speichern von Datensätzen benutzt wird. Man kann sich die Mehrfachdefinition von Typen sparen, wenn diese die gleichen Eigenschaften besitzen.

```
typedef itk::ImageFileWriter<InternalImageType>
   InternalWriterType;
```

Anschließend kann wieder eine neue Instanz des neuen Typs erstellt werden. Diese bekommt den Input mittels `SetInput()` und einen Dateinamen mittels `SetFileName()` übergeben, unter dem gespeichert werden soll.

```
InternalWriterType::Pointer writer =
   InternalWriterType::New();
writer -> SetInput(filter -> GetOutput());
writer -> SetFileName(Filename);
writer -> Update();
```

Wiederum mit dem Aufruf von `Update()` wird die Ausführung des `writers` gestartet.

4.1.3 Caster

Teilweise ist es in einer Funktion nötig den Datentyp eines Volumens zu ändern. Beispielsweise nach der Filterung des Volumens wird der Datensatz von

InputImageType nach InternalImageType gecastet. Hierfür stellt ITK die Klasse CastImageFilter zur Verfügung, mit deren Hilfe ein Datensatz von einem Datentyp in einen anderen Datentyp konvertiert werden kann.

Auch hier muss zuerst wieder die entsprechende Headerdatei inkludiert werden, damit die Klasse zur Verfügung steht.

```
#include "itkCastImageFilter.h"
```

Danach kann dann wieder ein Typ der Klasse definiert werden, in dem angegeben wird von welchem Datentyp in welchen konvertiert werden soll. In diesem Fall wird von InputImageType in den InternalImageType umgewandelt.

```
typedef  itk::CastImageFilter<
              InputImageType,
              InternalImageType>
   InputToInternalCastImageFilterType;
```

Hat man den Typ definiert, kann eine neue Instanz des Typs erstellt werden, welcher man dann den Datensatz vom reader mittels SetInput() übergibt.

```
InputToInternalCastImageFilterType::Pointer cast =
   InputToInternalCastImageFilterType::New();
cast -> SetInput(reader -> GetOutput());
cast -> Update();
```

Zum Schluss wird der caster durch den Aufruf Update() gestartet.

4.1.4 ITK nach VTK konvertieren

Da ITK Daten nicht direkt an VTK übergeben kann, müssen hierfür die Daten zuerst umgewandelt werden. Diese Aufgabe übernimmt die Funktion ConnectPipelines(), der man den itkExporter und den vtkImporter übergibt.

```
ExportFilterType::Pointer itkExporter =
      ExportFilterType::New();

itkExporter->SetInput( reader->GetOutput() );

vtkImageImport* vtkImporter = vtkImageImport::New();

ConnectPipelines(itkExporter, vtkImporter);

vtkImporter->Update();
```

4.2 Spezifische Funktionen

Nachdem nun die grundlegenden Funktionen, wie einen Datensatz einlesen und speichern, beschrieben wurden, soll nun mit der Erläuterung der spezifischen Funktionen begonnen werden, in denen die einzelnen Filter zur Bildverarbeitung und Visualisierung erklärt werden. Es werden dabei, falls vorhanden, verschiedene Filter vorgestellt, welche ähnliche Aufgaben erfüllen und letztendlich begründet, welcher Filter aus welchen Gründen am besten für die Anwendung geeignet ist.

4.2.1 Datensätze einlesen

In der Medizin liegen Datensätze aus CT- beziehungsweise MR-Geräten im DICOM-Format vor, welches zu Beginn in den Grundlagen 2.2.3 näher erläutert wurde. Somit war es das Hauptanliegen dieses Format zu unterstützen, um keine unnötigen Schritte über eine Konvertierung vom DICOM-Format in ein anderes Format vornehmen zu müssen. Für den Fall, dass doch ein anderes Format vorliegt, wird noch zusätzlich zum DICOM-Format das RAW-Format unterstützt. Es ist ein gängiges Format, welches auch 3D-Datensätze speichern kann.
Im gleichen Arbeitsschritt wird direkt nach dem Einlesen des Datensatzes dieser auch gleich mit einem Filter geglättet, um Rauschen aus dem Volumen zu entfernen und bessere Grundlagen für die Segmentierung zu schaffen.
Danach kann dann der Datensatz für die nächsten Schritte mit dem normalen `InternalWriterType` gespeichert werden.

4.2.1.1 DICOM Datensatz

ITK stellt speziell für DICOM-Datensätze verschiedene DICOM-Funktionen zur Verfügung. Da es in dieser Arbeit um 3D-Volumendaten geht und DICOM-Dateien meist als 2D-Schichtbilder vorliegen, benötigt man die Möglichkeit, mehrere Schichtbilder zu einem Volumenbild einzulesen. Das Einlesen von Bilderserien zu einem Volumen wird in ITK durch den `ImageSeriesReader` umgesetzt. Dieser bekommt die Dateinamen der einzulesenden Schichtbilder und deren Typ übergeben. Um allerdings die Dateinamen übergeben zu können müssen diese erst einmal bekannt sein. Dieses Problem wird durch den Filter `DICOMSeriesFileNames` gelöst. Er bekommt das Verzeichnis mit den darin enthaltenen Schichtbildern übergeben und ermittelt, welche Dateien im DICOM-Format darin vorliegen und leitet deren Namen an den darauf folgenden `ImageSeriesReader` weiter. Der `SeriesReader` liest die Schichtbilder darauf einzeln ein und setzt aus allen Einzelbildern ein großes Volumenbild zusammen.

Zu Beginn muss wiederum die Headerdatei eingebunden werden, damit man auf die Klasse `itkDICOMSeriesFileNames` `zugreifen kann.`

```
#include "itkDICOMSeriesFileNames.h"
```

Danach wird dann wieder eine neue Instanz erzeugt und ihr das Verzeichnis übergeben.

```
itk::DICOMSeriesFileNames::Pointer nameGenerator =
      itk::DICOMSeriesFileNames::New();
nameGenerator->SetDirectory( folder );
```

In diesem Verzeichnis wird dann nach der DICOM-Serie gesucht, die fortlaufend nummeriert sein sollte, und dann zum Schluss vom `nameGenerator` mittels `GetFileNames` im Array `fileNames` zurückgegeben.

```
typedef vector< string> seriesIdContainer;
const seriesIdContainer & seriesUID =
      nameGenerator->GetSeriesUIDs();

seriesIdContainer::const_iterator seriesItr =
      seriesUID.begin();
seriesIdContainer::const_iterator seriesEnd =
      seriesUID.end();

typedef vector<string> fileNamesContainer;
fileNamesContainer fileNames;

char inputFilenames[2];
fileNames = nameGenerator->GetFileNames( inputFilenames );
```

Die so erzeugten Dateinamen werden dann an den `ImageSeriesReader` übergeben, der nach dem gleichen Prinzip arbeitet wie der Standard-Reader. Der Unterschied ist jedoch, dass er zum einen mehrere Bilder zu einem Volumen einliest und noch den Typ der eingelesenen Bilder mitgeteilt bekommt, wodurch er weiß, wie er mit ihnen umzugehen hat. In diesem Fall wird ihm die Information übergeben, dass es sich bei der Bilderserie um DICOM-Bilder handelt. Aufgrund dieser Information weiß der Reader nun, dass die Dateien der übergebenen Dateinamen zusätzlich zu den normalen Bildinformationen einen bestimmten Header besitzen, der beim Einlesen der Bildinformationen übersprungen werden muss.

Wie immer müssen zu Beginn die benötigten Headerdateien, in diesem Fall für den `ImageSeriesReader` und die DICOMImageIO eingebunden werden.

```
#include "itkImageSeriesReader.h"
#include "itkDICOMImageIO2.h"
```

Danach können dann wieder neue Typen definiert und davon wiederum neue Instanzen erzeugt werden, die für diesen Schritt notwendig sind. Wir benötigen einen `reader` vom Typ `ImageSeriesReader` und die Typangabe `dicomIO`, um sagen zu können von welchem Typ die Bilder sind. Da die

Grauwerte der DICOM-Bilder im Folgenden durch den Glättungsfilter laufen müssen und dieser `float`-Zahlen benötigt, wird als Datentyp zum Einlesen direkt schon `float` im `InputImageType` verwendet.

```
typedef itk::ImageSeriesReader<InputImageType>
      InputReaderType;
SliceReaderType::Pointer reader = SliceReaderType::New();
itk::DICOMImageIO2::Pointer dicomIO =
      itk::DICOMImageIO2::New();
```

Hat man diese erzeugt, kann man dem Reader die Dateinamen und den Typ der Serienbilder übergeben.

```
reader->SetFileNames( fileNames );
reader->SetImageIO( dicomIO );
reader->Update();
```

Wie immer wird mittels `Update()` der Reader gestartet.

4.2.1.2 RAW Datensatz

Wie schon erwähnt, wird zusätzlich zum Einlesen von DICOM-Datensätzen, auch das Einlesen von RAW-Datensätzen unterstützt, um nicht nur an das DICOM-Format gebunden zu sein. Allerdings benötigen RAW-Datensätze Zusatzinformationen über beispielsweise die Dimension und die Ausdehnung des Datensatzes, die an das Programm übergeben werden müssen. Diese müssen bei diesem Format bekannt sein, da ohne diese Angaben der Datensatz unbrauchbar wäre. Meist liegen diese Informationen in einer zweiten Textdatei dem eigentlichen Datensatz bei.

Zu Beginn müssen wieder die benötigten Headerdateien eingebunden werden.

```
#include "itkRawImageIO.h"
#include "itkImageFileReader.h"
```

Sind diese bekannt kann man sich wieder einen `ReaderType` definieren und davon eine neue Instanz erstellen. Zusätzlich benötigt man wieder den Typ des einzulesenden Bildes, den man an das Programm übergeben muss, damit dieses weiß, wie es mit den Bildinformationen umzugehen hat.

```
typedef itk::ImageFileReader<InputImageType>
      RAWReaderType;
RAWReaderType::Pointer reader = RAWReaderType::New();
RawImageIOType::Pointer rawImageIO=RawImageIOType::New();
```

Danach kann damit begonnen werden dem Reader die benötigten Informationen zu übergeben, die er für das Einlesen der RAW-Datei benötigt. Neben dem Dateinamen und dem Typ des Bildes werden hier unter anderem die Dimension, der Ursprung, die Ausdehnung des Volumens in jede Richtung und die Ausdehnung eines Voxels in jede Richtung angegeben. Mit diesen Informationen kann dann der `RawReader` das Volumen korrekt einlesen und weitergeben.

```
reader->SetFileName(inputFilename);
reader->SetImageIO(rawImageIO);

rawImageIO->SetByteOrderToBigEndian();
rawImageIO->SetFileTypeToBinary();
rawImageIO->SetFileDimensionality(3);

rawImageIO->SetOrigin(0,0.0);
rawImageIO->SetOrigin(1,0.0);
rawImageIO->SetOrigin(2,0.0);

rawImageIO->SetDimensions(0,dimensionX);
rawImageIO->SetDimensions(1,dimensionY);
rawImageIO->SetDimensions(2,dimensionZ);

rawImageIO->SetSpacing(0,spacingX);
rawImageIO->SetSpacing(1,spacingY);
rawImageIO->SetSpacing(2,spacingZ);

reader->Update();
```

Durch die `Update()` Funktion wird wiederum der `reader` dazu veranlasst das angegebene Bild mit den Zusatzinformationen einzulesen.

4.2.1.3 Datensatz Filtern

Um den Datensatz für die weitere Verarbeitung vorzubereiten und ihn von Rauschen zu befreien muss er, wie in den Grundlagen 2.3.2 schon erklärt wurde, gefiltert werden. Da eine normale Filterung mittels eines linearen Filters zu viele Konturen verwischen würde, die für eine saubere Segmentierung benötigt werden, wird auf nicht-lineare Filter zurückgegriffen, welche die Kanten erhalten und trotzdem das Rauschen in Flächen unterdrücken. ITK bietet hierzu mehrere Glättungsfilter an: Den `GradientAnisotropicDiffusionImage-Filter`, den `CurvatureFlowImageFilter` oder den `BilateralImageFilter` und Abwandlungen von diesen, die alle eine Art von kanten-erhaltendem Glätten unterstützen.

- **GradientAnisotropicDiffusionImageFilter**

  Bei diesem Filter wird, wie im Paper „Scale-Space and Edge Detection Using Anisotropic Diffusion“ von Perona und Malik [1] erläutert wird, der Grad der Glättung durch eine Diffusionsfunktion berechnet, die vom Gradienten des aktuell betrachteten Voxels abhängt. Im Randbereich von Flächen nimmt die Funktion einen geringen Wert an, wodurch die Glättung minimiert bis sogar gestoppt wird und dadurch die Kanten erhalten bleiben. In den Flächen an sich findet allerdings eine normale Glättung der Artefakte statt. Somit wird eine kanten-erhaltende Glättung und unter Umständen sogar eine Schärfung der Konturen erreicht.

- **CurvatureFlowImageFilter**

  Beim `CurvatureFlowImageFilter` wird wiederum mit Hilfe einer Diffusionsfunktion der Grad der Glättung berechnet. Kanten mit gleichen Grauwerten bilden dabei entsprechende LevelSets, die an die Diffusionsgleichung übergeben werden. Kanten mit starken Krümmungen werden hierbei stärker geglättet, als Kanten mit schwächerer Krümmung, was zu einem Verschwinden kleinerer Artefakte und gleichzeitig zu einem gewissen Informationsverlust führen kann. Trotzdem können andere Artefakte übrig bleiben, die man eigentlich geglättet haben wollte.

- **BilateralImageFilter**

  Der bilaterale Filter ist, wie bei C. Tomasi [3] beschrieben, ein nichtlinearer, kanten-erhaltender Glättungsfilter, welcher sich aus einem linearen Gaussfilter und einer nicht-linearen Funktion zusammensetzt. Es können damit sowohl Grauwertbilder, als auch Farbbilder gefiltert werden. Bei der Berechnung fließen sowohl die geometrische Nähe, als auch deren Farbwerte ein. Die Gewichtungen der Nachbarschaften des Gaussfilters sind dabei abhängig vom Abstand zum Zentralpixel. Um einer Kantenverwischung entgegenzuwirken, wird eine Strafffunktion definiert, welche einen zu großen Abstand bei Kanten durch Vergleiche der Intensitäts- und Farbunterschiede kompensiert.
  Durch bilaterale Filterung medizinischer Bilddaten kann es allerdings zu einer Verwischung feiner Strukturen kommen. Dem kann durch Anpassung verschiedener Parameter und eine größere Anzahl an Wiederholungen entgegengewirkt werden, was allerdings die Rechenzeit vergrößert.
  Wilbur C.K. Wong beschreibt in seinem Paper „Trilateral Filtering for Biomedical Images“ [4] ein ähnliches aber überarbeitetes Verfahren, welches weniger Wiederholungen benötigt, um gute Ergebnisse zu erzielen. Dieser Filter ist allerdings in ITK noch nicht umgesetzt.

Die Wahl des Filters fiel auf den `GradientAnisotropicDiffusion-ImageFilter`, der wie die klassische Perona-Malik-Diffusion [1] arbeitet. Dieser wird ebenfalls in der Arbeit von Sherbondy [2] als kantenerhaltender Glättungsfilter empfohlen und angewandt. Der Filter erhält die Kanten, indem die Glättung durch den Filter an den Kanten verlangsamt oder gar gestoppt wird. Der Filter hat im Gegensatz zum `CurvatureFlowImageFilter` oder `BilateralImageFilter` den Vorteil, dass kleine Strukturen nicht verschwinden, sondern eher noch geschärft werden.

Zu Beginn benötigt man wieder die Headerdatei, in der der Filter näher beschrieben wird.

```
#include "itkGradientAnisotropicDiffusionImageFilter.h"
```

Daraus kann man sich dann wieder einen neuen Filtertyp definieren, aus dem man seine neue Filterinstanz erzeugt.

```
typedef itk::GradientAnisotropicDiffusionImageFilter<
            InputImageType,
            InputImageType >
            GradientAnisotropicDiffusionImageFilterType;;

GradientAnisotropicDiffusionImageFilterType::Pointer
smoothing =
      GradientAnisotropicDiffusionImageFilterType::New();
```

Danach kann dann der Filter das Volumen vom `reader` mittels SetInput() übergeben bekommen.

```
smoothing->SetInput(reader->GetOutput());
```

Als nächstes benötigt der Filter noch Angaben über die Anzahl der Wiederholungen die er ausführen soll, die Größe der `TimeSteps`, die angibt, wie stark in den Flächen geglättet werden soll, und schließlich die `Conductance`, die dafür zuständig ist, wie stark an den Kanten geglättet werden soll.

```
smoothing->SetNumberOfIterations( 2 );
smoothing->SetTimeStep( 0.125 );
smoothing->SetConductanceParameter( 1 );

smoothing->Update();
```

Durch die Funktion `Update()` wird der Filter wiederum gestartet.

4.2.2 Darstellung und Parameter wählen

Nachdem der erste Schritt nun abgeschlossen ist und das gefilterte Volumenbild vorliegt, können darin die Parameter der relevanten Strukturen gesucht werden, welche für die nächsten Schritte benötigt werden. Dies geht am besten mit einer Darstellung der einzelnen Schnittebenen.

Für die Visualisierung ist VTK zuständig. Es bietet hierzu die Möglichkeit der Darstellung mittels `vtkImagePlaneWidget`. Hierbei wird für jede der drei Raumachsen eine Schnittebene erzeugt, die dann einzeln verschoben und gedreht werden kann. Hat man die Ebenen so verschoben, dass man die zu untersuchende Struktur klar erkennt, können nun mittels eines Pickers [7.2] die Koordinaten und der Grauwert der Struktur bestimmt werden.

Das darzustellende Volumen wird zuerst von einem normalen ITK-Reader eingelesen und muss dann zur Darstellung in VTK mittels der zuvor beschriebenen Funktion `ConnectPipelines 4.1.4` an VTK übergeben werden.

Zu Beginn müssen Parameter wie die Größe des Datensatzes mit `GetSize()` und dessen Startpunkt mit `GetIndex()` ermittelt werden, um später die Größe der Ebenen und deren Position angeben zu können.

```
InternalImageType::Pointer inputImage = reader
      ->GetOutput();
InternalImageType::SizeType  size  =
      inputImage->GetBufferedRegion().GetSize();
InternalImageType::IndexType start =
      inputImage->GetBufferedRegion().GetIndex();
```

Danach kann der Picker mit seiner Toleranz und allgemeine Eigenschaften für die Beleuchtung und die Oberfläche geometrischer Objekte definiert werden, welche an die einzelnen Ebenen übergeben werden.

```
#include "vtkCellPicker.h"
#include "vtkProperty.h"

vtkCellPicker * picker = vtkCellPicker::New();
picker->SetTolerance(0.005);

vtkProperty * ipwProp = vtkProperty::New();
```

Im Folgenden wird nun beispielhaft an der X-Ebene demonstriert, wie die drei Schnittebenen definiert werden.
Es wird eine neue Instanz von `vtkImagePlaneWidget` erzeugt und ihr die entsprechenden Parameter, wie zum Beispiel das darzustellende Volumen, die Ausrichtung und die Position übergeben. Auch der zuvor definierte Picker wird hier eingebunden.

```
#include "vtkImagePlaneWidget.h"

vtkImagePlaneWidget * xImagePlaneWidget =
      vtkImagePlaneWidget::New();

xImagePlaneWidget->DisplayTextOn();
xImagePlaneWidget->SetInput(vtkImporter->GetOutput());
xImagePlaneWidget->SetPlaneOrientationToXAxes();
xImagePlaneWidget->SetSliceIndex(size[0]/2);
xImagePlaneWidget->SetPicker(picker);
xImagePlaneWidget->RestrictPlaneToVolumeOn();
xImagePlaneWidget->SetKeyPressActivationValue('x');
xImagePlaneWidget->GetPlaneProperty()->SetColor(1, 0, 0);
xImagePlaneWidget->SetTexturePlaneProperty(ipwProp);
xImagePlaneWidget
            ->SetResliceInterpolateToNearestNeighbour();
```

Weiter muss die eigentliche VTK-Darstellungs-Pipeline aufgebaut werden, die sich aus dem Renderer, dem Fenster und dem Interactor zusammensetzt. Hierzu werden Instanzen dieser erzeugt und der `renderer` dann an das `renderWindow` übergeben, das dann wieder an den

RenderWindowInteractor weitergegeben wird. Es werden zusätzlich noch Parameter wie Fenstergröße und Hintergrundfarbe festgelegt.

```
vtkRenderer* renderer = vtkRenderer::New();
vtkRenderWindow* renderWindow = vtkRenderWindow::New();
vtkRenderWindowInteractor* renderWindowInteractor =
      vtkRenderWindowInteractor::New();

renderWindow->SetSize(600, 600);
renderWindow->AddRenderer(renderer);
renderWindowInteractor->SetRenderWindow(renderWindow);
renderer->SetBackground( 0.1, 0.2, 0.4 );
```

Um dann mit den erstellten Ebenen interagieren zu können, wird mittels SetInteractor() der gerade erstellte renderWindowInteractor an die einzelnen Ebenen übergeben.

```
xImagePlaneWidget->SetInteractor( renderWindowInteractor );
xImagePlaneWidget->On();
```

Danach muss dann nur noch das renderWindow durch Render() zur Darstellung veranlasst und der renderWindowInteractor gestartet werden und man bekommt die Schnittebenen dargestellt, mit denen dann interagiert werden kann..

```
renderWindow->Render();
renderWindowInteractor->Start();
```

Damit nach der Darstellung die gerade benötigten Instanzen die Rechenzeit und den Speicher nicht weiter beeinträchtigen, müssen diese nach ihrem Gebrauch in VTK manuell gelöscht werden. Hierzu dient der Aufruf Delete().

```
xImagePlaneWidget->Delete();
picker->Delete();
ipwProp->Delete();
renderer->Delete();
renderWindow->Delete();
renderWindowInteractor->Delete();
```

4.2.3 Segmentierung

Im dritten Schritt, der Segmentierung, sollen nun die Blutbahn und das Skelett aus dem eingelesenen, gefilterten Volumen segmentiert werden. Da die Blutbahn neben den Verkalkungen das Wichtigste bei der späteren Darstellung und Diagnose ist, wird diese mit speziellen Algorithmen segmentiert. ITK bietet zur Segmentierung eine Vielzahl von Filtern, die auch im ITK Software Guide [9] näher erklärt werden.

- **Pixelorientierte Verfahren:**

  - **BinaryThresholdImageFilter:**

    Der Filter wandelt ein Grauwertbild in ein Binärbild um. Hierzu muss der Benutzer einen unteren und einen oberen Schwellwert, sowie einen Inside- und einen Outside-Wert angeben. Es wird jedes Voxel des Volumens mit dem oberen und unteren Schwellwert verglichen. Liegt der Grauwert dazwischen, wird der Inside-Wert zurückgegeben, andernfalls der Outside-Wert.

- **Regionenorientierte Verfahren:**

  - **ConnectedThresholdImageFilter:**

    Dieser Filter ist ähnlich wie der BinaryThresholdImageFilter ein Schwellwertfilter. Der Unterschied zwischen diesen beiden ist allerdings, dass der ConnectedThresholdImageFilter ein Region-Growing-Filter ist und zusätzlich zu den Schwellwerten einen Saatpunkt benötigt. Es werden hierbei auch nicht alle Voxel des Volumens bezüglich der Schwellwerte untersucht, sondern nur Nachbarvoxel des Saatpunktes. Fallen deren Grauwerte zwischen die Schwellwerte werden diese zum segmentierten Volumen hinzugefügt und wiederum deren Nachbarn bezüglich des Schwellwertes untersucht. Dies wird so lange fortgesetzt, bis kein Voxel mehr zum Volumen hinzugefügt werden kann. Der ConnectedThresholdImageFilter eignet sich sehr gut als Region-Growing-Filter für unhomogene Flächen, in denen auch kleine Grauwertunterschiede vorkommen. Der Nachteil ist, dass die Schwellwerte recht genau angegeben werden müssen. Das genaue Eingrenzen kann unter Umständen recht umständlich sein.

  - **ConfidenceConnectedImageFilter:**

    Dieser Filter berechnet den Durchschnitt und die Standardabweichung der Grauwerte aller Voxel der aktuellen Region um den Startpunkt herum. Ein vom Benutzer vorgegebener Wert wird mit der Standardabweichung multipliziert, um eine Region um den Durchschnittswert zu definieren. Zu der segmentierten Region werden dann ausgehend vom Saatpunkt diejenigen Voxel hinzugefügt, bei denen sich der Grauwert der Nachbarvoxel innerhalb dieser Region um den Durchschnitt herum befindet. Wurden alle Voxel, die die Bedingung erfüllen, dem Volumen hinzugefügt, ist der erste Durchlauf abgeschlossen und die Standardabweichung und der Durchschnitt können von diesen Voxeln neu berechnet werden. Mit diesen neu berechneten Werten startet dann der

zweite Durchlauf der Segmentierung. Dies wird nun anhand einer zuvor festgelegten Anzahl von Wiederholungen fortgesetzt, wobei in jedem Durchlaufe Voxel zum Volumen hinzugefügt werden können. Der `ConfidenceConnectedImageFilter` eignet sich somit zum Segmentieren homogener Flächen, in denen keine starken Grauwertunterschiede auftreten, beziehungsweise der Grauwertunterschied benachbarter Voxel gering ist.

- **Kantenorientierte Verfahren:**

  Kantenorientierte Verfahren benötigen klar definierte Kanten und werden hauptsächlich auf 2D-Bildern angewendet, um Objekte zu segmentieren. Sie kommen hier nicht zur Anwendung und werden deshalb nur kurz erwähnt.
  Die Filter benötigen mehrere Schritte wie beispielsweise eine Kantenextraktion, darauf folgend eine Kantenverdünnung und schließlich die Kantenverfolgung, um geschlossene Konturen zu finden und zu segmentieren.

- **Texturorientierte Verfahren**

  Die texturorientierte Segmentierung wird ebenfalls nicht näher betrachtet, da sie hier keine Anwendung findet. Die zu segmentierenden Flächen weisen größtenteils einheitliche Grauwerte auf, was einfacher durch die zuvor genannten Filter zu lösen ist.

Im Folgenden wird nun die Implementierung der verschiedenen Segmentierungsverfahren, welche zum Einsatz kamen, näher erläutert.

Für die Segmentierung der Blutbahn eignen sich am besten die regionenorienterten Segmentierungsverfahren, da diese zusammenhängende Strukturen zurückliefern und auf Grauwertähnlichkeiten basieren. Aus diesem Grund wurden auch beide Verfahren, die ITK zur Verfügung stellt, implementiert, um sie testen zu können.

Das Skelett hingegen, das im gleichen Schritt mitsegmentiert werden soll, wird nur grob mittels des pixelorientierten Verfahrens segmentiert. Es dient später nur der groben Orientierung im Körper und bedarf deshalb keiner genaueren Segmentierung.

4.2.3.1 Blutbahn – ConnectedThresholdImageFilter

Der `ConnectedThresholdImageFilter`, den ITK zur Verfügung stellt, benötigt als Eingabe nur einen Saatpunkt und einen Schwellwertbereich für die Segmentierung der Blutbahn. Das Problem dabei liegt in der Wahl eines geeigneten Schwellwertbereichs, damit der Filter nicht zu viel vom Umfeld mitsegmentiert oder vielleicht dazu gehörende Bestandteile weglässt. Hier kommt es auf eine gewissenhafte Datenerhebung im Vorfeld an.

Zu Beginn muss wie immer die Headerdatei des Filters eingefügt werden.

```
#include "itkConnectedThresholdImageFilter.h"
```

Danach kann dann wieder ein neuer Typ des Filters definiert und davon eine neue Instanz erzeugt werden.

```
typedef itk::ConnectedThresholdImageFilter<
               InternalImageType,
               SegmentedImageType >
                                       ConnectedFilterType;
ConnectedFilterType::Pointer connectedThreshold =
   ConnectedFilterType::New();
```

Diesem neuen Filter werden dann mittels `SetInput()` das Volumen vom `reader`, die Schwellwerte mittels `SetLower()` und `SetUpper()` und ein Farbwert, auf den das segmentierte Volumen gesetzt werden soll, übergeben.

```
connectedThreshold->SetInput( reader->GetOutput() );
connectedThreshold->SetLower( lowerThreshold );
connectedThreshold->SetUpper( upperThreshold );
connectedThreshold->SetReplaceValue( replaceValue );
```

Danach fehlt noch der Saatpunkt, von dem die Segmentierung gestartet werden soll, der dem Filter mittels `SetSeed()` übergeben wird.

```
InternalImageType::IndexType  index;
index[0] = seedX;
index[1] = seedY;
index[2] = seedZ;
connectedThreshold->SetSeed( index );
```

Wie immer wird der Filter mittels `Update()` gestartet.

```
connectedThreshold->Update();
```

Als Ergebnis erhält man die segmentierte Blutbahn als Binärbild, bei dem die Farbwerte der Blutbahn auf den vorgegebenen Grauwert gesetzt wurden und der Rest des Bildes den Grauwert „0“ hat. Da nur zwei Grauwerte im Bereich zwischen „0“ und „255“ vorkommen genügt es das segmentierte Volumen unter dem Datentyp „`char`“ zu speichern, um Speicherplatz zu sparen und dadurch die Performanz beim Speichern und Einlesen in den folgenden Schritten zu erhöhen.

4.2.3.2 Blutbahn – ConfidenceConnectedImageFilter

Die zweite Möglichkeit zur Segmentierung, die implementiert wurde, ist der `ConfidenceConnectedImageFilter`, der wie in (0) schon erwähnt wurde, allerdings eher für homogene Flächen geeignet ist.

Zu Beginn wird wieder die Headerdatei des Filters inkludiert.

```
#include "itkConfidenceConnectedImageFilter.h"
```

Danach wird wiederum ein neuer Typ des Filters definiert und eine Instanz von ihm erzeugt.

```
typedef itk::ConfidenceConnectedImageFilter<
              InternalImageType,
              SegmentedImageType >
                    ConfidenceConnectedImageFilterType;

ConfidenceConnectedImageFilterType::Pointer confidence =
   ConfidenceConnectedImageFilterType::New();
```

Als nächstes können dem Filter die benötigten Parameter übergeben werden. So wird ihm mittels `SetInput()` das zu filternde Volumen vom `reader` übergeben und mittels `SetMultiplier()` der Wert mit dem die Standardabweichung multipliziert wird. Weiter bekommt er noch die Anzahl der Wiederholungen, die Nachbarschaftsgröße und den Grauwert, der in das neue Volumen eingetragen wird, übergeben.

```
confidence->SetInput( reader->GetOutput() );
confidence->SetMultiplier( multiplier );
confidence->SetNumberOfIterations( numberOfIterations );
confidence->SetInitialNeighborhoodRadius( neighborhood );
confidence->SetReplaceValue( replaceValue );
```

Als letztes fehlt nur noch der Saatpunkt, der definiert und mittels `SetSeed()` an den Filter übergeben wird.

```
typedef ConfidenceConnectedImageFilterType::IndexType
   IndexType;
IndexType seed;

seed[0] = ( seedX );
seed[1] = ( seedY );
seed[2] = ( seedZ );

confidence->SetSeed( seed );

confidence->Update();
```

Wie üblich wird der Filter mittels `Update()` gestartet.

Als Ausgabe bekommt man bei geeigneter Parameterwahl und homogenen Grauwerten des zu segmentierenden Objektes die segmentierte Blutbahn als Binärbild zurückgeliefert. Diese wird dann unter dem Datentyp „`char`“ gespeichert.

4.2.3.3 Skelett – BinaryThreshold

Da das Skelett wie schon erwähnt nur der groben Orientierung im Körper dient, ist eine genaue Segmentierung mittels regionenorientiertem Verfahren hier nicht nötig und wäre sogar übertrieben, da sich das Skelett aus einer Vielzahl einzelner Knochen zusammensetzt. Es müsste somit für jeden Knochen ein Saatpunkt gesetzt werden, damit diese dann auch segmentiert würde. Die einfachste Lösung für diese Segmentierung bietet der erwähnte `BinaryThresholdImageFilter`, der alle Voxel des Volumens aufgrund der Schwellwerte hin untersucht und gegebenenfalls zum Segment hinzufügt.

Zu Beginn muss wiederum die Headerdatei des Filters inkludiert werden.

```
#include "itkBinaryThresholdImageFilter.h"
```

Danach kann dann wieder ein neuer Typ des Filters definiert und eine neue Instanz von diesem erzeugt werden.

```
typedef itk::BinaryThresholdImageFilter<
               InternalImageType,
               SegmentedImageType >
                     BinaryThresholdImageFilterType;

BinaryThresholdImageFilterType::Pointer binaryFilter =
   BinaryThresholdImageFilterType::New();
```

Dieser neuen Instanz werden dann wieder die benötigten Parameter übergeben. Zuerst wird wieder mittels `SetInput()` das Volumen vom `reader`, gefolgt von den Schwellwerten und den Werten, auf die das segmentierte Skelett beziehungsweise das Umfeld gesetzt werden sollen, übergeben.

```
binaryFilter->SetInput( reader->GetOutput() );

binaryFilter->SetLowerThreshold( lowerThresholdSkelett );
binaryFilter->SetUpperThreshold( upperThresholdSkelett );
binaryFilter->SetOutsideValue( outsideValueSkelett );
binaryFilter->SetInsideValue( insideValueSkelett );

binaryFilter->Update();
```

Zum Abschluss folgt wie immer der Aufruf von `Update()`, um den Filter zu starten.

Die Ausgabe dieses Filters ist wieder ein Binärbild und kann Speicherplatz sparend unter dem Datentyp „`char`“ gespeichert werden, ohne dass Informationen verloren gehen.

4.2.4 Region of Interest

Den nächsten Schritt im Ablauf bildet die Erstellung von zwei *Regions of Interest*. Dies ist nicht unbedingt notwendig, beschleunigt aber den Ablauf der nächsten Schritte ungemein, da nicht auf den kompletten Volumen weitergearbeitet werden muss, sondern nur auf Teilvolumen.
ITK bietet hierzu den `RegionOfInterestImageFilter` an, der ein neues Bild in der gleichen Dimension wie das Originalbild erzeugt, in dem allerdings nur die durch einen Startpunkt und Ausdehnung angegebene Region abgebildet ist. Diese Punkte hat man sich im zweiten Schritt bei der Suche nach den Parametern gewählt, indem man die einzelnen Ebenen zu Recht geschoben hat.

Zu Beginn wird wieder die Headerdatei eingebunden.

```
#include "itkRegionOfInterestImageFilter.h"
```

Danach wieder wiederum ein neuer Typ des Filters definiert und eine neue Instanz von diesem erzeugt.

```
typedef itk::RegionOfInterestImageFilter<
               InternalImageType,
               InternalImageType >

                     RegionOfInterestImageFilterType;

RegionOfInterestImageFilterType::Pointer regionFilter =
   RegionOfInterestImageFilterType::New();
```

Als nächstes werden dann der Startpunkt und die Ausdehnung der Region angegeben und als `desiredRegion` mittels `SetRegionOfInterest` an den Filter übergeben.

```
InternalImageType::IndexType start;
start[0] = startX ;
start[1] = startY ;
start[2] = startZ ;

InternalImageType::SizeType size;
size[0] = sizeX ;
size[1] = sizeY ;
size[2] = sizeZ ;

InternalImageType::RegionType desiredRegion;
desiredRegion.SetSize( size );
desiredRegion.SetIndex( start );

regionFilter->SetRegionOfInterest( desiredRegion );
```

Wie immer bekommt der Filter mittels `SetInput()` das Volumen, mit dem er arbeiten soll, vom `reader` übergeben.

```
regionFilter->SetInput( reader->GetOutput());

regionFilter->Update();
```

Mittels `Update()` wird dann auch wieder der Filter gestartet.

Der Filter liefert die zu Beginn beschriebene *Region of Interest* zurück. Da für die nächsten Schritte sowohl mit der Blutbahn als auch dem originalen Volumen gearbeitet wird, wird von beiden Volumen eine *Region of Interest* erzeugt und unter den Datentyp „`signed short`" gespeichert, da schließlich alle Grauwerte wie im originalen Volumen erhalten bleiben sollen.

4.2.5 DistanceMap

Der nächste Schritt wird benötigt, um im weiteren Verlauf die Möglichkeit zu haben, nur im Umfeld der segmentierten Region nach Verkalkungen suchen zu können. Es wird mittels der ITK-Klasse `DanielssonDistanceMapImageFilter` eine Entfernungskarte (*DistanceMap*) erstellt, in der festgehalten ist, wie groß der euklidsche Abstand jedes Voxels des Volumens zur segmentierten Blutbahn ist.

Um den Filter benutzen zu können, muss wiederum zuerst die Headerdatei eingebunden werden.

```
#include "itkDanielssonDistanceMapImageFilter.h"
```

Danach kann dann wieder ein neuer Typ des Filters definiert und eine neue Instanz erzeugt werden.

```
typedef itk::DanielssonDistanceMapImageFilter<
                SegmentedImageType,
                InternalImageType >

   DanielssonDistanceMapImageFilterType;

DanielssonDistanceMapImageFilterType::Pointer filter =
   DanielssonDistanceMapImageFilterType::New();
```

Danach braucht nur noch das Volumen vom `reader` an den Filter mittels `SetInput()` übergeben und dem Filter gesagt werden, von welchem Typ das ihm übergebene Bild ist. In diesem Fall handelt es sich um die segmentierte Blutbahn, also ein Binärbild.

```
filter->SetInput( reader->GetOutput());
filter->InputIsBinaryOn();

filter->Update();
```

Zum Schluss wird dann der Filter mittels `Update()` gestartet.

Der Filter liefert ein Bild zurück, in dem die Entfernungen zur segmentierten Blutbahn eingetragen sind. Die segmentierte Blutbahn an sich hat die Entfernung 0, direkt angrenzende Voxel den Abstand 1 und so weiter.

4.2.6 Verkalkungen extrahieren

Als letzter Schritt vor der Darstellung fehlt die Extrahierung der Verkalkungen. Da eine Selektion der einzelnen Kalkflecken zu aufwändig wäre, benötigt man ein Verfahren, um die Verkalkungen automatisch zu extrahieren. Dies wird dadurch gelöst, dass man die Verkalkungen aufgrund ihrer Grauwerte ausfindig macht. Diese Grauwerte kann man sich im zweiten Schritt der Darstellung und Parametersuche wählen oder erfahrungsgemäß einfach den Grauwertbereich für Kalk benutzen. Hierbei besteht das Problem, dass man nicht den ganzen Körper durchsuchen darf, sondern nur den Bereich, in dem die gesuchten Verkalkungen auftreten können, also in direkter Nähe der Blutbahn. Würde man im kompletten Datensatz suchen, würde man als Ergebnis auch noch zusätzlich zu den Verkalkungen das Skelett zurückgeliefert bekommen. Um die Region um die Blutbahn einschränken zu können, wurde der Zwischenschritt über die *DistanceMap* gemacht, wodurch man nun eine beliebige Region mit einem bestimmten Abstand zur Blutbahn auswählen kann.

Diese Region erhält man durch einen einfachen Binärfilter, der schon bei der Extrahierung des Skelettes benutzt wurde. Diesem übergibt man einen Schwellwertbereich, sowie die *DistanceMap*, in der die Entfernungen der einzelnen Voxel zur Blutbahn eingetragen sind. In diesem Fall soll der Bereich vom Nachbarvoxel der Blutbahn, mit einem Abstand von „1“, bis zu einem angemessenen Abstand darum herum gewählt werden. Meist genügt ein geringer Abstand von bis zu 3 Voxeln, um darin die Verkalkungen sauber extrahieren zu können. Die in diesem Bereich liegenden Voxel bekommen den Binärwert „1“ zugewiesen, alle übrigen den Wert „0“. Dies ist für den nächsten Schritt von Bedeutung.

In ITK wird das ganze umgesetzt, indem man wieder zuerst die Headerdatei des Filters einbindet.

```
#include "itkBinaryThresholdImageFilter.h"
```

Danach kann dann wieder ein neuer Typ des Filters definiert und eine neue Instanz erzeugt werden.

```
typedef itk::BinaryThresholdImageFilter<
             InternalImageType,
             SegmentedImageType >
                  BinaryThresholdImageFilterType;

BinaryThresholdImageFilterType::Pointer binaryFilter1 =
   BinaryThresholdImageFilterType::New();
```

Diesem Filter wird mittels `SetInput()` die *DistanceMap* vom `reader` übergeben. Danach werden dann mittels `SetLowerThreshold()` und `SetUpperThreshold()` die Schwellwerte angegeben, in denen sich die Region befindet. Zu letzt werden noch die Werte benötigt, auf welche die entsprechenden Regionen gesetzt werden sollen.

```
binaryFilter1->SetInput( reader2->GetOutput());

binaryFilter1->SetLowerThreshold( lowerThresholdUmfeld );
binaryFilter1->SetUpperThreshold( upperThresholdUmfeld );

binaryFilter1->SetOutsideValue( outsideValueUmfeld );
binaryFilter1->SetInsideValue(  insideValueUmfeld  );

binaryFilter1->Update();
```

Zum Schluss wird wie immer der Filter mittels `Update()` gestartet.

Als Ergebnis bekommt man ein Volumen, in dem eine Region im angegebenen Abstand um die Blutbahn herum auf einen Wert von „1“ und der Rest des Volumens auf den Wert „0“ gesetzt wurde.
Da es sich wieder um ein Binärbild handelt, genügt es mit dem Datentyp `char` weiter zu arbeiten, um Speicher zu sparen.

Bisher hat man allerdings noch nicht die originalen Grauwerte mit den darin enthaltenen Verkalkungen. Diese erhält man, indem man das gerade erzeugte Volumen mit dem originalen Volumen multipliziert. Dadurch, dass das nahe Umfeld der Blutbahn den Wert „1“ hat und der Rest den Wert „0“, bekommt man bei der Multiplikation der beiden Volumen ein Volumen zurück, in dem die originalen Grauwerte des Umfeldes enthalten sind und der Rest durch die Multiplikation mit „0“ auf „0“ gesetzt wurde.

ITK bietet hierzu einen einfachen Multiplikationsfilter `MultiplyImageFilter`, der zwei zu multiplizierende Volumen übergeben bekommt.

Wie immer wird zu Beginn die Headerdatei eingebunden.

```
#include "itkMultiplyImageFilter.h"
```

Danach kann dann wieder ein neuer Typ des Filters definiert und eine neue Instanz erzeugt werden.

```
typedef itk::MultiplyImageFilter<
               InternalImageType,
               SegmentedImageType,
               InternalImageType >
                     MultiplyImageFilterType;

MultiplyImageFilterType::Pointer multiplyFilter =
```

```
      MultiplyImageFilterType::New();
```

Als nächstes werden dann die beiden zu multiplizierenden Volumen vom `reader` beziehungsweise dem `binaryFilter1` an den Filter mittels `SetInput1()` und `SetInput2()` übergeben und danach der Filter mittels `Update()` gestartet.

```
multiplyFilter->SetInput1( reader1->GetOutput());
multiplyFilter->SetInput2( binaryFilter1->GetOutput());

multiplyFilter->Update();
```

Als Ausgabe bekommt man die originalen Grauwerte der Voxel, die an die Blutbahn angrenzen. Als Datentyp für dieses Volumen wird wiederum `signed short` benutzt, um den Erhalt aller Graustufen sicherzustellen.

In diesem Volumen kann dann die eigentliche Extrahierung der Verkalkungen stattfinden, indem man mit dem Binärfilter die Grauwerte der Verkalkungen herausfiltert.

Hierzu wird wieder die Headerdatei des Binärfilters benötigt.

```
#include "itkBinaryThresholdImageFilter.h"
```

Von dieser Klasse kann dann wiederum ein neuer Typ definiert und eine neue Instanz erzeugt werden.

```
typedef itk::BinaryThresholdImageFilter<
               InternalImageType,
               SegmentedImageType >
                     BinaryThresholdImageFilterType;

BinaryThresholdImageFilterType::Pointer binaryFilter2 =
   BinaryThresholdImageFilterType::New();
```

Wie zuvor bekommt der Filter mittels `SetInput()` das Volumen vom `multiplyFilter`, den Grauwertbereich der Verkalkungen und die Werte, auf welche die einzelnen Regionen gesetzt werden sollen, übergeben.

```
binaryFilter2->SetInput( multiplyFilter->GetOutput() );

binaryFilter2->SetOutsideValue( outsideValueKalk );
binaryFilter2->SetInsideValue( insideValueKalk );

binaryFilter2->SetLowerThreshold( lowerThresholdKalk );
binaryFilter2->SetUpperThreshold( upperThresholdKalk );

binaryFilter2->Update();
```

Nach dem Start des Filters durch `Update()` bekommt man die Verkalkungen, deren Grauwerte im Schwellwertbereich liegen, als Binärbild zurückgeliefert. Diese können dann wieder unter dem Datentyp `char` gespeichert werden.

4.2.7 Darstellung

Nachdem nun alle zur computergestützten Diagnose notwendigen Volumen vorliegen, kann mit der Darstellung mittels VTK begonnen werden. Hierzu müssen die drei Datensätze zuerst vorbereitet werden, bevor sie an den `renderer` übergeben werden können. Dies wird beispielhaft für die Blutbahn demonstriert, ist allerdings bei den anderen beiden Volumen synonym zu handhaben.
Als erstes muss aus dem 3D Volumendatensatz, der nur aus Voxeln mit Farbwerten besteht, eine Oberfläche erzeugt werden. Diese Aufgabe übernimmt der `vtkContourFilter`. Danach wird die Oberfläche mittels des `vtkSmoothPolyDataFilter` geglättet und durch den Filter `vtkPolyDataNormals` die Normalen des Polygonnetzes berechnet. Der Filter `vtkStripper` macht dann zusätzlich aus allen Polygonen durch Triangulierung Dreiecke, die daraufhin durch den Filter `vtkPolyDataMapper` in grafische Primitiven umgewandelt werden.

Zu Beginn jedes Filters wird die benötigte Headerdatei eingebunden und daraus eine neue Instanz der Klasse erzeugt.

```
#include "vtkContourFilter.h"

vtkContourFilter *contour1 = vtkContourFilter::New();
```

Danach werden dem `ContourFilter` das Volumen und der Grauwert übergeben, für den eine Oberfläche generiert werden soll.

```
contour1->SetInput( reader1->GetOutput() );
contour1->SetValue( 0, contourValue );
```

Die erzeugte Oberfläche wird dann an den Glättungsfilter `vtkSmoothPolyDataFilter` weitergegeben. Dieser Filter glättet die Oberfläche, indem er einen Laplace-Weichzeichner verwendet. Dabei wird jeder Eckpunkt auf den Durchschnittswert seiner Nachbareckpunkte gesetzt. Die zwei wichtigsten Parameter sind die Anzahl der Wiederholungen, die angeben, wie oft der Filter über die Oberfläche laufen soll, und die Konvergenz, die angibt, wie weit sich ein Punkt bewegen darf. Bewegt er sich weniger als die Konvergenz, wird der Prozess gestoppt.

Von diesem Filter muss erst wieder einmal eine neue Instanz erzeugt werden.

```
#include "vtkSmoothPolyDataFilter.h"

vtkSmoothPolyDataFilter *smooth1 =
   vtkSmoothPolyDataFilter::New();
```

Nachdem die `contour` übergeben wurde, kann man dem Filter die Anzahl der Wiederholungen und die Konvergenz übergeben und die komplette Szene mittels `UpdateWholeExtent()` aktualisieren.

```
smooth1->SetInput( contour1->GetOutput() );
smooth1->SetNumberOfIterations(100);
smooth1->SetConvergence(0);

smooth1->UpdateWholeExtent();
```

Um später die Objekte richtig beleuchten zu können, ist es notwendig, die Normalen zu erzeugen. Dies wird mit dem Filter `vtkPolyDataNormals` erreicht, der aus den Oberflächennormalen der einzelnen Polygone an deren Eckpunkten den Durchschnitt bildet und dadurch die Eckpunktnormalen berechnet.

Es muss wieder zu Beginn eine neue Instanz des Filters erzeugt werden.

```
#include "vtkPolyDataNormals.h"

vtkPolyDataNormals *normals1 = vtkPolyDataNormals::New();
```

Danach können dann dem Filter die geglätteten Oberflächen vom Weichzeichner übergeben werden. Weiter ist es möglich dem Filter zu sagen, ab welcher Gradzahl es sich bei einem Winkel um einen spitzen Winkel handelt, damit der Filter dem entgegenwirken kann und somit keine leuchtenden Eckpunkte entstehen.

```
normals1->SetInput(smooth1->GetOutput());

normals1->SetFeatureAngle(60.0);
```

Der nächste Filter `vtkStripper` berechnet alle in den Oberflächen vorkommenden Dreiecke und gibt sie einzeln zurück.
Nachdem eine neue Instanz des Filters erzeugt wurde, wird ihm einfach die Oberfläche, für die er die Dreiecke berechnen soll, übergeben.

```
#include "vtkStripper.h"

vtkStripper *stripper1 = vtkStripper::New();

stripper1->SetInput(normals1->GetOutput());
```

Zum Abschluss des ersten Teils der VTK-Pipeline folgt der `vtkPolyDataMapper`. Dieser bildet die Polygondaten auf grafische Primitive ab, die dann im zweiten Teil der Pipeline visualisiert werden können.

```
#include "vtkPolyDataMapper.h"

vtkPolyDataMapper *polyDataMapper1 = vtkPolyDataMapper::New();
```

Nachdem wieder eine neue Instanz des Filters erzeugt wurde, können ihr die Oberflächen übergeben werden. Mittels der Methode `ScalarVisibilityOff()` wird dem Filter mitgeteilt, dass er nicht die Skalare benutzen soll, um die Objekte einzufärben. Dies findet erst im nächsten Schritt statt, indem jedem `actor` eine Farbe gegeben wird.

```
polyDataMapper1->SetInput( stripper1->GetOutput());

polyDataMapper1->ScalarVisibilityOff();
```

Danach kann mit dem zweiten Teil der VTK-Pipeline begonnen werden, in dem die Objekte dargestellt werden. Als erstes werden die Objekte an die `actoren` übergeben, welche die Objekte repräsentieren und diese an den `renderer` weitergegeben. Die Objekte werden nun vom `renderer` an das `renderWindow` weitergereicht, wo sie nun final dargestellt werden. Um mit den dargestellten Volumen noch interagieren zu können wird ein `renderWindowInteractor` benötigt, der die Eingaben über Tastatur und Maus koordiniert.

Zu Beginn wird wieder die Headerdatei eingebunden und eine neue Instanz des `actors` erzeugt.

```
#include "vtkActor.h"

vtkActor *actor1 = vtkActor::New();
```

Dieser Instanz kann dann der `polyDataMapper` mittels `SetMapper()` übergeben werden, welcher das Objekt enthält, das der `actor` in der Szene repräsentieren soll. Zusätzlich können dem `actor` noch weitere Eigenschaften wie Farbe und Transparenz zugewiesen werden.

```
actor1->SetMapper( polyDataMapper1 );

actor1->GetProperty()->SetColor(1,0,0);
actor1->GetProperty()->SetOpacity(opacityBlutbahn);
```

Hat man diese Filter auf die drei Volumen angewendet, hat man schließlich drei `actoren` vorliegen und kann diese an den `renderer` mittels AddActor() übergeben. Dem `renderer` wird schließlich zur kontrastreichen Darstellung noch eine Hintergrundfarbe zugewiesen, die sich gut von den Farben der einzelnen `actoren` abhebt.

```
#include "vtkRenderer.h"

vtkRenderer * renderer = vtkRenderer::New();

renderer->AddActor( actor3 );
renderer->AddActor( actor1 );
renderer->AddActor( actor2 );

renderer->SetBackground( 0.1, 0.2, 0.4 );
```

Um bei der Visualisierung mit den dargestellten Objekten interagieren zu können, wird noch der `vtkRenderWindowInteractor` benötigt. Nachdem seine Headerdatei eingebunden ist, kann eine neue Instanz erzeugt und der Interactor mittels `Start()` gestartet werden.

```
#include "vtkRenderWindowInteractor.h"

vtkRenderWindowInteractor * renderWindowInteractor =
   vtkRenderWindowInteractor::New();

renderWindowInteractor->Start();
```

Um schließlich etwas auf dem Display sehen zu können, muss noch das `RenderWindow` erzeugt werden.

```
#include "vtkRenderWindow.h"

vtkRenderWindow * renderWindow = vtkRenderWindow::New();
```

Diesem wird eine Fenstergröße mittels `SetSize()`, der `renderWindowInteractor` mittels `SetInteractor()` und der `renderer` mittels `AddRenderer()` zugewiesen. Mit dem Befehl `Render()` wird schließlich die Darstellung gestartet.

```
renderWindow->SetSize(600, 600);
renderWindow->SetInteractor( renderWindowInteractor );
renderWindow->AddRenderer( renderer );

renderWindow->Render();
```

Ganz zum Schluss muss noch sichergestellt werden, dass die von VTK erzeugten Instanzen wieder mittels `Delete()` gelöscht werden, um den Speicher nach beendeter Darstellung nicht unnötig zu belasten.

```
renderer->Delete();
renderWindow->Delete();
renderWindowInteractor->Delete();

contour1->Delete();
normals1->Delete();
stripper1->Delete();
actor1->Delete();
polyDataMapper1->Delete();
```

4.3 Benutzeroberfläche

Zur Erstellung einer Benutzeroberfläche wurde, wie bereits in den Grundlagen 2.4.3 erwähnt, das Toolkit FLTK benutzt. FLTK bietet eine gute Anbindung an ITK und VTK und eignet sich somit gut zur Eingabe von Parametern und zur Visualisierung. Die Eingabe der Parameter über eine Benutzeroberfläche hat im Gegensatz zur Eingabe über eine Konsole den Vorteil, dass der Benutzer immer alle Parameter für den jeweiligen Schritt im Auge hat und diese gegebenenfalls auch individuell anpassen kann.

4.3.1 Aufbau

Beim Aufbau wurde speziell darauf geachtet, jeden Schritt getrennt von den anderen darzustellen. Mit Hilfe der Umsetzung durch Karteikarten ist es möglich, dem Benutzer gleichzeitig den Ablauf der einzelnen Schritte vorgibt. Hierdurch bekommt der Benutzer die sieben Schritte in chronologischer Reihenfolge präsentiert, so wie er sie für einen kompletten Durchlauf des Programms benötigt. Darüber hinaus wird ihm die Möglichkeit geboten auch Schritte einzeln oder in anderer Reihenfolge auszuführen. Dies ist nur möglich, da in jedem Schritt neu eingelesen und gespeichert wird. Benötigt wird dies, um Zwischenergebnisse kontrollieren zu können, indem beispielsweise die segmentierte Blutbahn aus Schritt 3 im Picker anzeigen werden kann lässt.

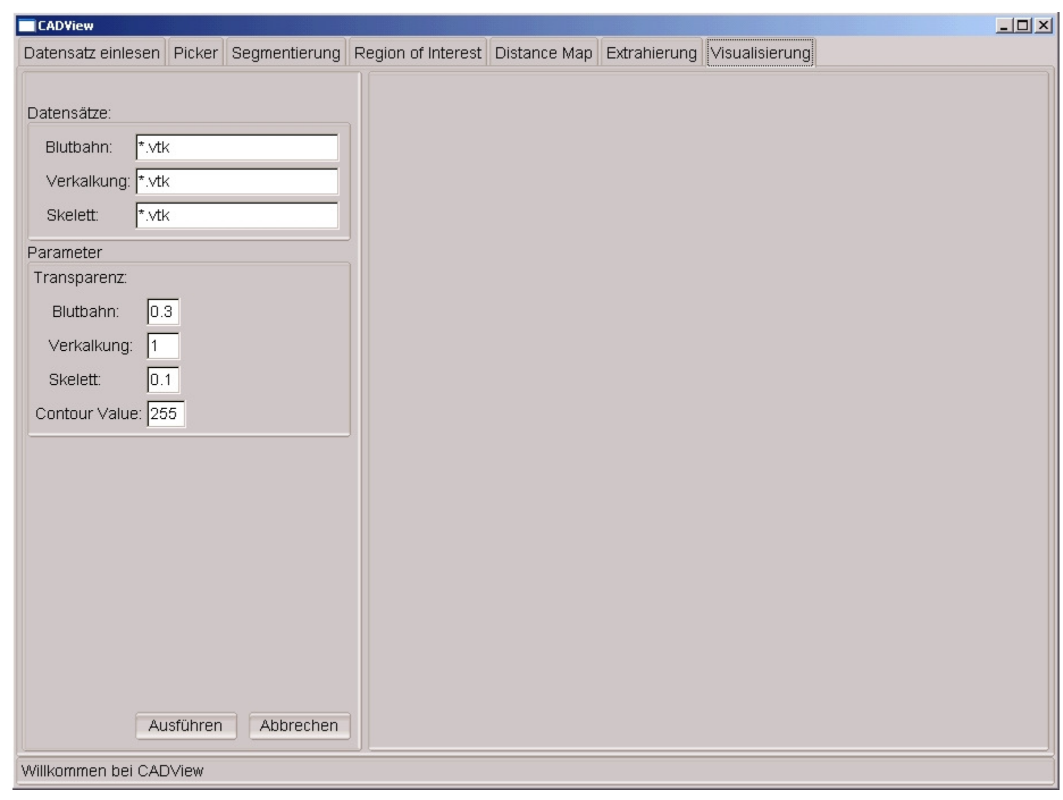


Abbildung 19: Benutzeroberfläche

Um die Benutzeroberfläche möglichst konsistent zu halten, wurde zuerst ein grobes Gerüst erstellt, in das die einzelnen Funktionen eingebettet wurden. So sind immer am oberen Rand die sieben Reiter der Hauptschritte zu sehen, damit man jederzeit zwischen den Schritten wechseln kann. Darunter folgt dann das Hauptfeld, das senkrecht in zwei kleine Gruppen unterteilt wurde.
In der linken Gruppe hat der Benutzer die Möglichkeit, wie beispielsweise im Schritt „Datensatz einlesen“, zwischen zwei Reitern zu wählen. Je nachdem von welcher Art sein Datensatz ist, kann er sich für das Einlesen eines DICOM- oder eines RAW-Datensatzes entscheiden. Darunter folgen dann die Eingabefelder für die gewählte Funktion.
In der rechten, größeren Gruppe ist in den Schritten „Picker“ und „Visualisierung“ Platz für ein Fenster, in dem die Volumen visualisiert werden und wo man mit ihnen interagieren kann. In allen anderen Schritten bleibt diese Gruppe leer.
Den Abschluss bildet unterhalb des Hauptfeldes eine Fußzeile.

Um dem Benutzer die Bedienung zu erleichtern, wurden weitere Felder, die in jeder Funktion auftauchen, wie Dateinamen zum Einlesen oder Speichern, immer an der gleichen Stelle positioniert. Weiter wurde darauf geachtet, zusammengehörige Parameter auch visuell zusammen zu gruppieren. So wurden beispielsweise beim Sammeln der Parameter im „Picker“ die Parameter für die Segmentierung und die Parameter für die *Regions of Interest* getrennt zu Gruppen zusammengefasst und umrandet. Weiter ist jeder Gruppe beziehungsweise den Textfeldern ein Tooltip zugewiesen, der beim Überfahren des Objektes mit der Maus Informationen über die benötigte Eingabe ausgibt.

4.3.2 Erstellung einer Benutzeroberfläche

Zur einfachen Erstellung einer Benutzeroberfläche bietet FLTK ein kleines Programm namens „Fluid“. Es ist eine grafische Benutzeroberfläche mit der es dem Benutzer ermöglicht wird, seine Benutzeroberfläche individuell zusammenzustellen, Parameter zu setzen und Funktionen zuzuweisen. Dabei ist hauptsächlich auf eine richtige Verschachtelung und Gruppierung der Objekte zu achten. Die Objekte, die verwendet werden um die einzelnen Funktionen zu gruppieren, sind Hauptfenster, Tabs und Gruppen. Weiter wurden zur Eingabe von Zahlen spezielle Textfelder für Zahlen und zur Eingabe von Texten wie den Dateinamen spezielle Textfelder für Buchstaben gewählt, die mit einem Label versehen werden können. Um Funktionen zu starten, beziehungsweise die Anwendung abzubrechen wurden normale Textbuttons verwendet.

Hat man sich die Benutzeroberfläche nach seinen Wünschen zusammengestellt, bietet Fluid die Möglichkeit, sich den C++-Code ausgeben zu lassen. Dieser kann dann einfach in die Main-Methode der eigenen Anwendung eingefügt werden.

Den Buttons kann dann mittels einer `Callback`-Zuweisung eine Funktion zugewiesen werden, die bei einem Klick darauf gestartet werden soll.

```
button->callback(funktionsname,0);
```

In den Funktionen an sich werden dann die Parameter der Textfelder mittels `->value()` ausgelesen und den entsprechenden Variablen zugewiesen.

```
int seedX =(segm_conf_seed_x->value());
```

Gleichzeitig werden hier direkt die Parameter an die darauf folgenden Funktionen übergeben, um dem Benutzer deren Eingabe abzunehmen.

```
roi_filename -> value(segm_ filename -> value());
```

5 Auswertung

Im nun folgenden Kapitel soll untersucht werden, wie gut sich die gewählten Filter für die entsprechende Aufgabe bewährt haben und welche Einstellungen nötig sind, um die bestmöglichen Ergebnisse zu erzielen. Um Vergleiche aufstellen zu können, wie gut die einzelnen Filter gearbeitet haben, werden Vergleichsvolumen herangezogen, an denen der Erfolg sichtbar wird.

Neben der Qualität der einzelnen Filter soll auch deren Zeitaufwand dokumentiert werden. Hierbei ist zu beachten, dass die Berechnungen auf einem Notebook mit einem mobile AMD Athlon 4 Prozessor mit 1200 MHz und 512 MB Arbeitsspeicher ausgeführt wurden. Eine verbaute Grafikkarte besaß keine 3D-Beschleunigung.

Zur Ermittlung der Dauer jeden Schrittes wurde vom Programm eine Ausgabe der Startzeit jeden einzelnen Schrittes erzeugt, so dass dessen Zeit berechnet werden konnte.

Zum Testen der entwickelten Anwendung CADView standen zwei Datensätze im DICOM-Format zur Verfügung, die beide einen Ausschnitt des Abdomens abbilden. Der erste Datensatz „Abdomen1“ besteht aus 228 Schichtbildern mit einem Speichervolumen von 144 MB. Der zweite Datensatz „Abdomen2“ sogar aus 358 Schichtbildern und einem Speichervolumen von 179 MB. Beide Datensätze besitzen eine Auflösung von 512 x 512 Pixel pro Schicht und sind sehr wahrscheinlich in einem Abstand von 1 mm aufgenommen worden.
Gearbeitet wurde primär mit dem Datensatz Abdomen1. Der zweite Datensatz wurde meist nur herangezogen, um zu prüfen, inwieweit die Verfahren auf andere Datensätze mit anderen Parametern anwendbar sind. Daraus ergeben sich natürlich auch Vergleiche bezüglich der Qualität der Anwendung in unterschiedlichen Datensätzen.

Im Folgenden wird nun genau auf das Vorgehen und die Ergebnisse der einzelnen Schritte der erstellten Anwendung CADView eingegangen.

5.1 Einlesen und Filtern

Der erste Teil des ersten Schrittes bildet das Einlesen des Datensatzes. Hierbei sind außer der Angabe des Verzeichnisses beziehungsweise des Dateinamens des Datensatzes keine weiteren Angaben zu machen. Insofern sind dieser Vorgang und dessen Ergebnis auch nicht weiter beeinflussbar. Der einzige Faktor, von dem der Einlesevorgang abhängt, ist die Größe des Datensatzes. Diese wirkt sich in der Dauer des Einlesens aus. Beide Testdatensätze werden in ungefähr einer Minute eingelesen und stehen für die weitere Verarbeitung zur Verfügung. Je kleiner natürlich die Datensätze sind, umso entsprechend geringer ist auch die

Zeit. Ein Ausschnitt aus den vorliegenden Datensätzen aus beispielsweise nur 5 Schichtbildern benötigt nur wenige Sekunden bis dieser zur Verfügung steht.
Den zweiten Teil des Einlesens bildet die Filterung, die direkt im Anschluss erfolgt. Die Filterung ist im Gegensatz zum reinen Einlesen sehr von Parametern abhängig und auch rechenzeitintensiver. Der kantenerhaltende Filter benötigt drei Parameter (4.2.1.3). Zum einen ist das die Anzahl der Wiederholungen. Je häufiger die Glättung durchgeführt werden soll, umso stärker wird das Bild auch geglättet, aber umso länger dauert sie logischerweise auch. Der nächste Parameter ist die Größe der *TimeSteps*. Dieser Wert ist dafür verantwortlich, wie stark der Filter glätten soll. Je größer dieser Wert ist, umso stärker glättet der Filter. Hierbei ist allerdings zu beachten, dass der Wert für die *TimeSteps* bei einem 3D-Bild nicht höher sein sollte als 0.125, um die Stabilität des Filters zu gewährleisten. Der dritte Parameter, die *Conductance*, ist zuständig für die Glättung an den Kanten, welche durch Anwendung eines kantenerhaltenden Filters erhalten bleiben sollen. Hier werden die Kanten umso mehr geglättet, je größer der Wert ist. Bei einem Wert von 0 findet keine Kantenglättung und bei größeren Werten eine entsprechend größere Kantenglättung statt.
Als gute Parameter haben sich die Werte „0.125“ für die *TimeSteps* und „1“ für die *Conductance* bei „2“ Wiederholungen herausgestellt. Somit hat man eine möglichst starke Glättung der Flächen mit einem guten Erhalt der Kanten bei einer recht niedrigen Wiederholungsanzahl.
Doch trotz der niedrigen Anzahl an Wiederholungen benötigt der Filter auf dem ersten Beispielvolumen knapp eine Stunde, um das Volumen zu filtern und dann zu speichern. Für den zweiten Beispieldatensatz wird noch ein größerer Zeitraum benötigt, da dieser Datensatz eine größere Anzahl von Schichtbilder beinhaltet. Der Filter benötigt zur Bearbeitung dieses Datensatzes annähernd 1,5 Stunden.
Als Ergebnis der Filterung erhält man allerdings bei beiden Beispieldatensätzen ein Volumen, bei dem die Bildstörungen in großen Flächen eindeutig reduziert wurden und Kanten trotzdem erhalten geblieben sind, wie es in den folgenden Bildern in Abbildung 20 zu sehen ist. Die Bilder sind Aufnahmen des Datensatzes „Abdomen1“. Der Filter liefert allerdings auf dem Datensatz „Abdomen2“ ähnlich gute Ergebnisse.
Die linken Bilder zeigen das ungefilterte Volumen. Auf den rechten ist im Vergleich dazu der gleiche Bildausschnitt zu sehen, nur eben gefiltert. Die Filterung ist vor allem in den grauen Flächen rechts und links der Wirbelsäule sehr gut zu erkennen. Hier sind im ungefilterten Bild noch recht viele Grauwertunterschiede zu erkennen, die im rechten Bild verschwunden sind. Weiter erkennt man gut, dass im originalen Volumen vorhandene Kanten immer noch sauber im gefilterten Bild zu erkennen sind. Die Kanten wurden somit, wenn überhaupt, nur sehr schwach geglättet beziehungsweise unschärfer gezeichnet.
Generell ist somit festzustellen, dass der gewählte kantenerhaltende Filter `GradientAnisotropicDiffusion` sehr gut seine Aufgabe erfüllt und das gewünschte Ergebnis zurückgeliefert hat. Es liegen nun von Rauschen befreite

Flächen und trotzdem noch sauber erkennbare Kanten vor, was für die nun folgenden Schritte notwendig ist.
Einziges Manko bildet der große Zeitaufwand der Filterung. Doch Versuche mit den anderen in der Umsetzung beschriebenen Filtern haben gezeigt, dass dort der Zeitaufwand genauso groß, wenn nicht sogar teilweise größer ist.

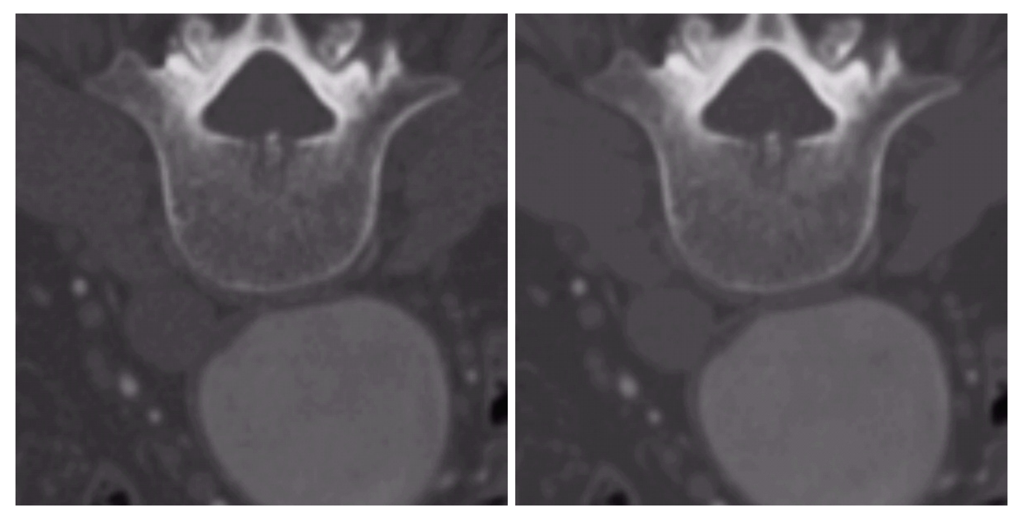


Abbildung 20: Vergleich ungefiltert/gefiltert - Schnitt Z-Achse

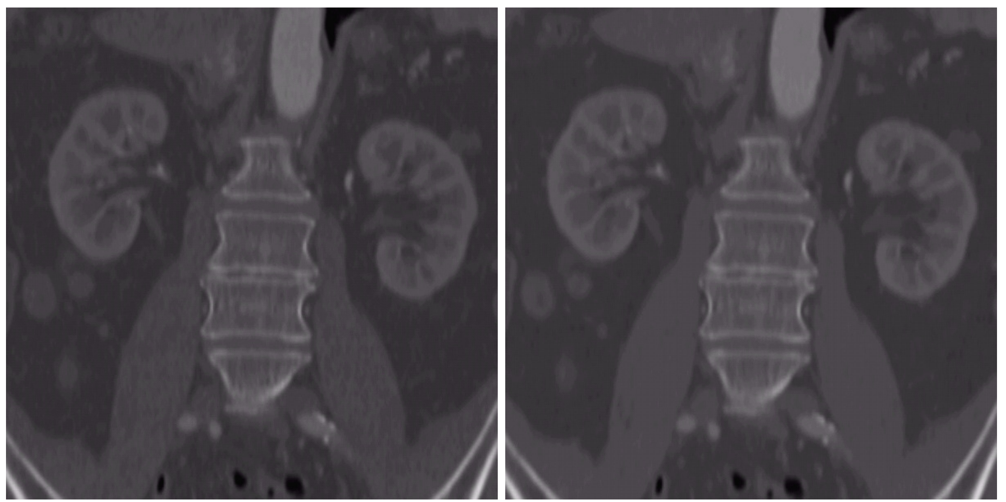


Abbildung 21: Vergleich ungefiltert/gefiltert - Schnitt Y-Achse

Aufgrund der extrem langen Zeit, die benötigt wird, um das Volumen zu filtern, kommt allerdings die Frage auf, ob sich die Filterung überhaupt lohnt. Mehrere Versuche mit dem ungefilterten Volumen haben gezeigt, dass wenn der Grauwertbereich bei der Segmentierung entsprechend gewählt ist, die Segmentierung der Blutbahn immer noch einwandfrei funktioniert. Voraussetzung dafür ist allerdings ein qualitativ sehr hochwertiger Datensatz, der an sich recht wenig Rauschen aufweist und mit einer geringen Schichtdicke aufgenommen wurde.

5.2 Picker

Im Picker werden die zu einem Volumen eingelesenen und gespeicherten Schichtbilder dargestellt, um darin Parameter für die folgenden Schritte zu wählen. Hierbei dauert das Einlesen des Datensatzes „Abdomen1“ mit einer Zeit von knapp zwei Minuten bei einem 150 MB großen Datensatz recht lange. Ist dieser erst einmal eingelesen wird er allerdings direkt von ITK nach VTK umgewandelt und dargestellt.
Die Darstellung ist dabei sehr performant hinsichtlich der Interaktion des Benutzers. So lässt sich das Volumen auch ohne 3D-Grafikkarte ruckelfrei im Raum drehen und skalieren, obwohl es vom Speichervolumen recht groß ist. Dies liegt daran, dass nur die drei Raumachsen und nicht das komplette Volumen dargestellt werden müssen.

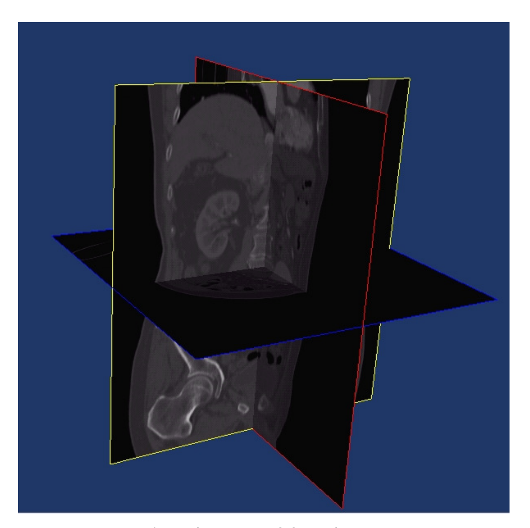


Abbildung 22: Picker

Die Darstellung in Form von den drei frei verschiebbaren Raumebenen hat sich als sehr vielseitiges Tool erwiesen. Man kann einfach und schnell anhand der Ebenen die zu segmentierenden Strukturen lokalisieren und deren Parameter erfassen und behält dabei den Überblick, an welcher Stelle man sich im Körper befindet.
Etwas aufwändiger ist die Wahl des Grauwertbereiches der Blutbahn, da hier die komplette Blutbahn manuell nach ihren Grauwerten abgesucht werden muss. Hier genügt allerdings oft eine grobe Eingrenzung des Bereiches.

Etwas Übung und Erfahrung benötigt man bei der Eingrenzung der *Region of Interest*, doch auch dies stellt keine allzu große Hürde dar, vor allem da es am Ergebnis nichts ändert, sollte die Region etwas zu groß gewählt sein, sondern sich gegebenenfalls nur die Rechenzeit etwas verlängert.

Weiter ist es nicht nur möglich im Picker die benötigten Parameter zu wählen, sondern auch Zwischenergebnisse schnell und einfach zu überprüfen, indem man sich die entsprechenden Volumen anzeigen lässt.

5.3 Segmentierung

Den nächsten Schritt im Programmablauf bildet die Segmentierung der Blutbahn und des Skelettes aus den Datensätzen.

5.3.1 Blutbahn

Eine gute Segmentierung der Blutbahn setzt einen guten Datensatz und eine korrekte Wahl der benötigten Parameter voraus. Bei sehr guten Ausgangsdatensätzen kann teilweise auf eine Glättung durch einen kantenerhaltenden Filter verzichtet werden. Doch die meisten Datensätze weisen relativ viel Rauschen auf und benötigen somit eine Glättung. Wie auch beim Picker wird bei der Segmentierung noch mit dem kompletten Datensatz gearbeitet und deshalb ist wiederum die Ladezeit des Volumens mit zwei Minuten recht lange. Die Segmentierung an sich ist mit wenigen Sekunden sehr schnell, was von der Größe des zu segmentierenden Objektes abhängig ist. Da die Blutbahn recht klein ist, fällt die benötigte Rechenzeit deshalb sehr kurz aus.

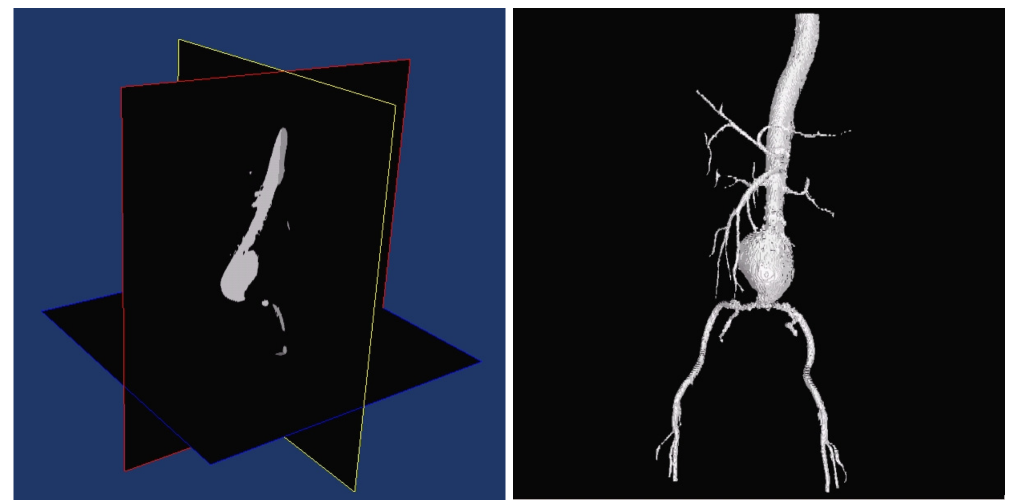


Abbildung 23: Segmentierte Blutbahn mittels Connected Threshold

Die Segmentierung mittels `ConnectedThresholdImageFilter` hat bei beiden Beispieldatensätzen die besten und zuverlässigsten Ergebnisse geliefert. So wird anhand der gewählten Parameter die komplette Blutbahn segmentiert (Abbildung 23). Sogar die rechte und linke Beinarterie und die Gefäße zu den Organen werden segmentiert.

Um allerdings zu zeigen, wie genau die Blutbahn segmentiert wurde, bedarf es eines Vergleiches zwischen dem originalen Volumen und der segmentierten Blutbahn. Hierzu wurde das Volumen der segmentierten Blutbahn des Datensatzes „Abdomen1“ invertiert und mit seinem originalen Volumen multipliziert. Als Ergebnis erhält man, wie in Abbildung 24 und Abbildung 25 zu sehen ist, ein Volumen, in dem die originalen Grauwerte vorhanden sind, bis auf die Voxel, die zu der segmentierten Blutbahn gehören. An diesen Stellen liegt ein Grauwert von „0“ vor.

In den linken Bildern sieht man das originale Volumen und erkennt die helle Blutbahn mit den weißen Verkalkungen am Rand. In den rechten Bildern hingegen wurde das originale Volumen mit der Blutbahn multipliziert, was durch eine Farbveränderung im Bereich der Blutbahn zu erkennen ist. Die komplette Blutbahn, bis auf einen kleinen Rand, wurde somit durch den `ConnectedThresholdImageFilter` segmentiert.

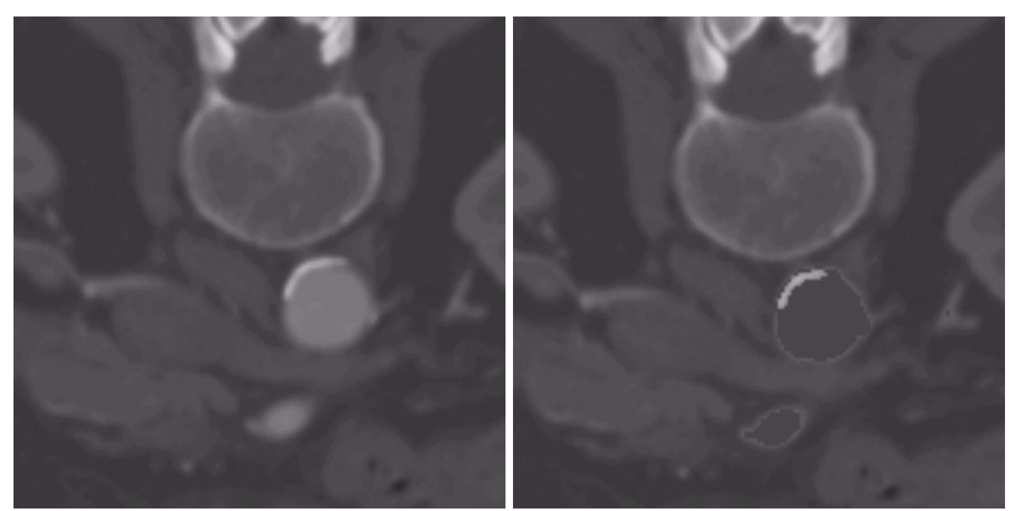


Abbildung 24: Vergleich Blutbahn - Schnitt Z-Achse

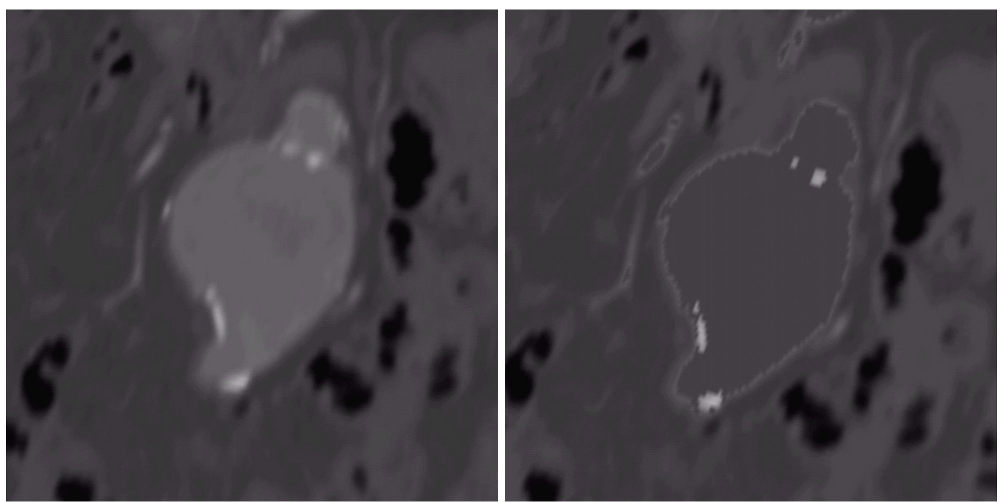


Abbildung 25: Vergleich Blutbahn - Schnitt Y-Achse

Bei der Segmentierung mittels des `ConfidenceConnectedImageFilter` traten hingegen beim Beispieldatensatz „Abdomen1“ meist Probleme bei der Segmentierung auf, da die Grauwerte der Blutbahn vom oberen bis zum unteren Ende recht unterschiedlich waren. Da dieses Verfahren auf der Ähnlichkeit der Grauwerte der benachbarten Voxel beruht, kommt es durch die Unterschiede zum Abbruch der Segmentierung, falls diese zu groß sein sollten. Andererseits trat häufiger das Problem auf, dass, wenn man das Maß für die Ähnlichkeit mittels Angabe einer höheren Farbabweichung verringerte, außer der Blutbahn noch weitere Strukturen, wie etwa das Skelett, mitsegmentiert wurden.

Die großen Grauwertunterschiede des Datensatzes „Abdomen1“ rühren wahrscheinlich von einer ungleichmäßigen Verteilung des Kontrastmittels her. Bei der Aufnahme ist somit auf eine gute Verteilung des Kontrastmittels zu achten, da es sonst zu Problemen bei der Segmentierung mittels des `ConfidenceConnectedImageFilter` kommen kann.

Der Beispieldatensatz „Abdomen2“ weist im Gegensatz zum Beispieldatensatz „Abdomen1“ so gut wie keine Farbunterschiede auf und ist recht homogen. Hier funktionierte die Segmentierung durch den `ConfidenceConnectedImageFilter` wie in Abbildung 26 zu sehen ist, einwandfrei. Voraussetzung ist allerdings die Homogenität der Grauwerte der Blutbahn. Um einen Fehler bei der Segmentierung auszuschließen, ist eine Segmentierung der Blutbahn mit diesem Filter nur bedingt zu empfehlen. Vor Anwendung muss sichergestellt werden, dass die Blutbahn vom oberen bis zum unteren Ende möglichst den gleichen Grauwert aufweist, da es sonst möglich ist, dass nur teilweise segmentiert wird.

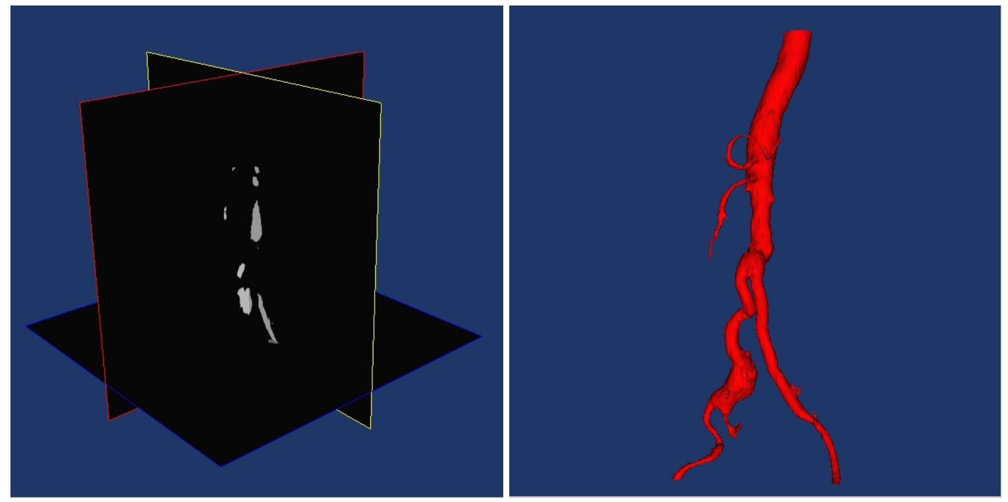


Abbildung 26: Segmentierte Blutbahn - Confidence Connected

5.3.2 Skelett

Genau wie bereits bei der Segmentierung der Blutbahn wird bei der Extrahierung des Skelettes auf dem kompletten originalen Datensatz gearbeitet. Um hierbei jedoch Ladezeit zu sparen, wird das Skelett im gleichen Schritt wie die Blutbahn segmentiert, so dass das originale Volumen nur ein Mal eingelesen werden muss. Die Extrahierung des Skelettes ist im Gegensatz zur Segmentierung der Blutbahn nur ein einfacher Schwellwertfilter, der die Farbwerte des Volumens mit einem Schwellwertbereich abgleicht und entsprechend dem extrahierten Volumen zuweist. Hierdurch ist kein großer Rechenaufwand nötig, wodurch auch hier sich die Rechenzeit auf wenige Sekunden beschränkt.
Von der Qualität ist der einfache Schwellwertfilter dem Segmentierungsalgorithmus klar unterlegen. Dies ist aber auch an dieser Stelle gewollt, da das Skelett keine wichtige Rolle bei der Diagnose spielt und eine

grobe Extrahierung ausreichend ist. Trotzdem ist das Skelett größtenteils zu erkennen, was für die Orientierung im Körper genügen sollte.

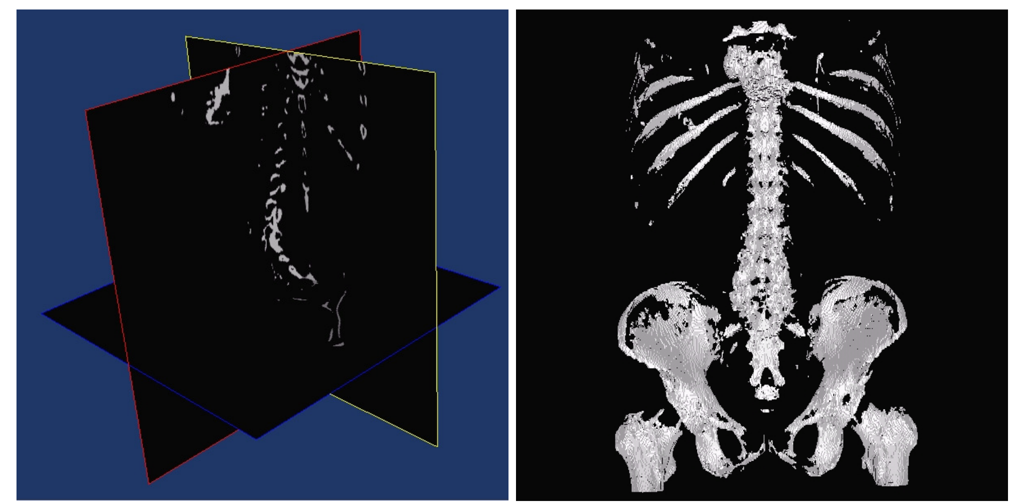


Abbildung 27: Segmentiertes Skelett

Der Schwellwertfilter ist einzig von dem ihm zugewiesenen Schwellwertbereich abhängig. Deshalb ist es notwendig möglichst alle Graustufen welche die Knochen aufweisen einzuschließen, um als Ausgabe einen Großteil des Skelettes zurückgeliefert zu bekommen. Für die eigentliche Diagnose der Verkalkungen spielt allerdings die Qualität der Extrahierung des Skelettes keine weitere Rolle.

5.4 Region of Interest

Die Erstellung und das Weiterarbeiten mit einer *Region of Interest* hat für das Ergebnisbild ebenfalls keine weitere Bedeutung, da keine relevanten Bildinformationen geändert werden, sondern nur die Größe beziehungsweise der Ausschnitt des Volumens mit dem gearbeitet wird, verändert wird. Doch hinsichtlich der Performanz ist ein deutlicher Unterschied zu spüren, ob mit dem originalen Volumen weitergearbeitet wird oder nur einem Teil davon. Gerade der nächste Schritt, die Erstellung der *DistanceMap*, benötigt auf dem originalen Datensatz über eine Stunde Rechenzeit. Angewandt auf die *Region of Interest* sind es nur wenige Minuten. Insofern rechnet sich der Zwischenschritt über die Erstellung der *Region of Interest* in zeitlicher Hinsicht enorm.

Die Erstellung der *Region of Interest* benötigte auf beiden Volumen nur wenige Minuten, wobei das Einlesen und Speichern der Volumen einen Großteil der in Anspruch genommenen Zeit ausmacht. Die eigentliche Reduzierung auf einen Ausschnitt des originalen Volumens beträgt nur wenige Sekunden.

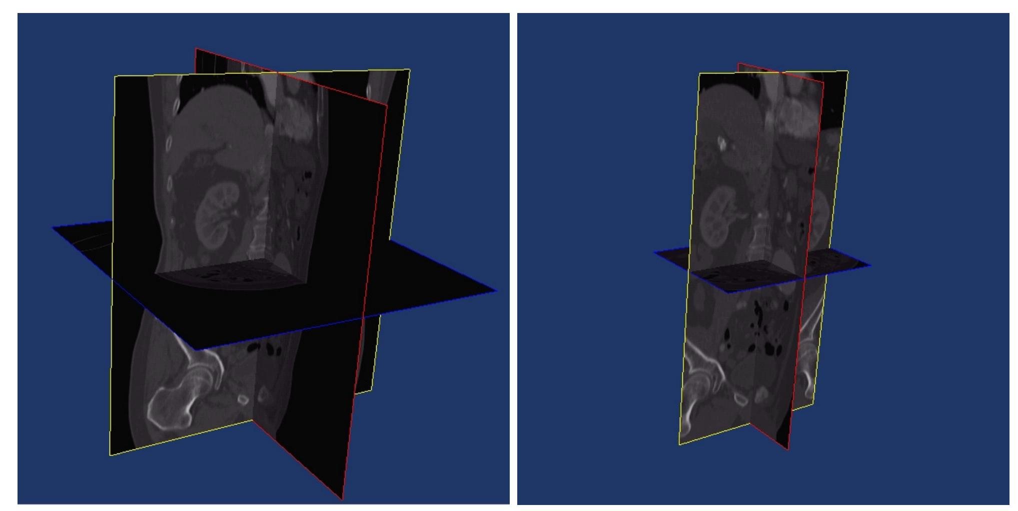


Abbildung 28: Region of Interest

5.5 DistanceMap

Die Erstellung einer *DistanceMap* ist nötig, um im nächsten Schritt, der Extrahierung der Verkalkungen, nur im nahen Umfeld der Blutbahn nach Verkalkungen zu suchen. Das Einlesen und Speichern der Volumen dauert hier nur wenige Sekunden, da mittlerweile nicht mehr mit dem originalen Volumen, sondern nur noch mit der *Region of Interest* gearbeitet wird. Die Berechnung der *DistanceMap* an sich dauert mit ungefähr 2:30 Minuten wiederum recht lange, benötigt aber im Vergleich zur Anwendung auf das originale Volumen nur einen Bruchteil der dafür benötigten Zeit.

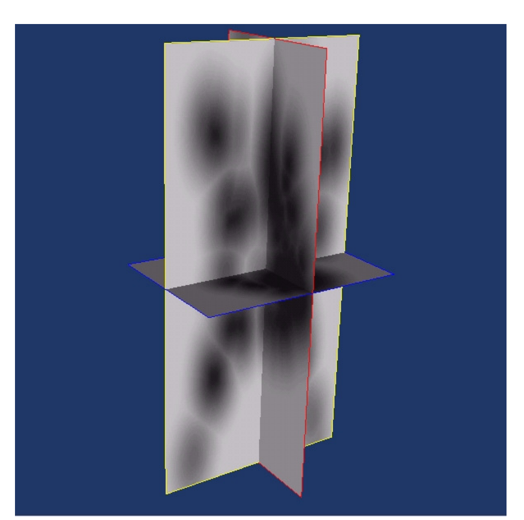


Abbildung 29: DistanceMap

Die Berechnung der *DistanceMap* benötigt viel Zeit, da recht viele Querbezüge hergestellt werden müssen, da die Blutbahn nicht nur aus der recht geradlinig verlaufenden Arterie besteht, sondern zusätzlich aus den kleineren Abzweigen zu den Organen beziehungsweise den Beinarterien.

5.6 Kalk Extrahierung

Die Extrahierung der Verkalkungen setzt sich, wie zuvor schon in der Umsetzung besprochen wurde, aus verschiedenen Schritten zusammen. So wird zuerst die *DistanceMap* binarisiert, indem ein Schwellwertfilter angewandt wird, der einen Bereich mit angegebenem Abstand um die Blutbahn herausfiltert und auf „1“ setzt. Darauf wird das Binärbild der Region um die Blutbahn mit dem originalen Volumen multipliziert. Als Ergebnis erhält man die originalen Grauwerte im Bereich um die Blutbahn. Der Rest des Volumens wurde dabei auf „0“ gesetzt. In diesem Volumen können nun wieder mittels eines Schwellwertfilters die Verkalkungen herausgefiltert werden.
Der komplette Arbeitsschritt dauert mit Einlesen, Binarisierung, Multiplikation, Extrahierung und Speicherung gerade einmal 40 Sekunden, was wieder auf die Größe des zu bearbeitenden Volumens zurückzuführen ist. Die Schritte Daten Einlesen und Speichern nehmen wiederum einen Großteil der benötigten Zeit in Anspruch. Die Schritte dazwischen spielen sich allerdings im Sekundenbereich ab.

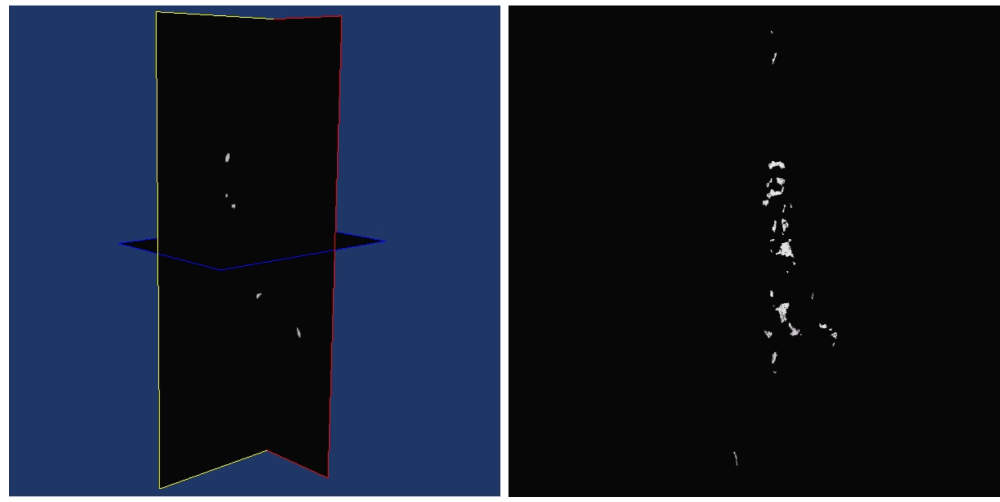


Abbildung 30: Extrahierte Verkalkungen

Dieses Verfahren ist zwar wiederum nur ein einfaches Schwellwertverfahren, doch durch das Wissen, dass Gefäßverkalkungen nur im nahen Umfeld von Gefäßen vorkommen können, kann die Filterung nach Verkalkungen auf genau diesen Bereich eingeschränkt werden. Würde man dies nicht machen, so würde man wie bei der Extrahierung des Skelettes alle Bereiche die dieselben Werte wie Verkalkungen haben, also auch die Knochen, mit extrahieren.
Betrachtet man das Ergebnis der Extrahierung wieder im Vergleich mit dem originalen Volumen (siehe Abbildung 31 und Abbildung 32), so fällt auf, dass fast alle Verkalkungen erfasst wurden. In den linken Bildern ist wiederum der schon bekannte Vergleich der segmentierten Blutbahn des Beispieldatensatzes „Abdomen1“ zu sehen. Gut erkennbar sind auch die noch vorhandenen weißen Verkalkungen an den Rändern der Blutbahn. Wird das Volumen mit den extrahierten Verkalkungen invertiert und mit dem vorherigen Vergleichsvolumen

multipliziert, so erhält man ein Volumen, in dem sowohl die Bereiche der segmentierten Blutbahn, als auch die der extrahierten Verkalkungen auf den Grauwert „0“ gesetzt wurden und der Rest noch den originalen Grauwert aufweist. Wäre ein Teil der Verkalkungen nicht extrahiert worden, so würde man in den rechten Bildern noch weiße Verkalkungen entdecken. Bis auf minimale Randstücke ist allerdings der komplette Kalk extrahiert worden.

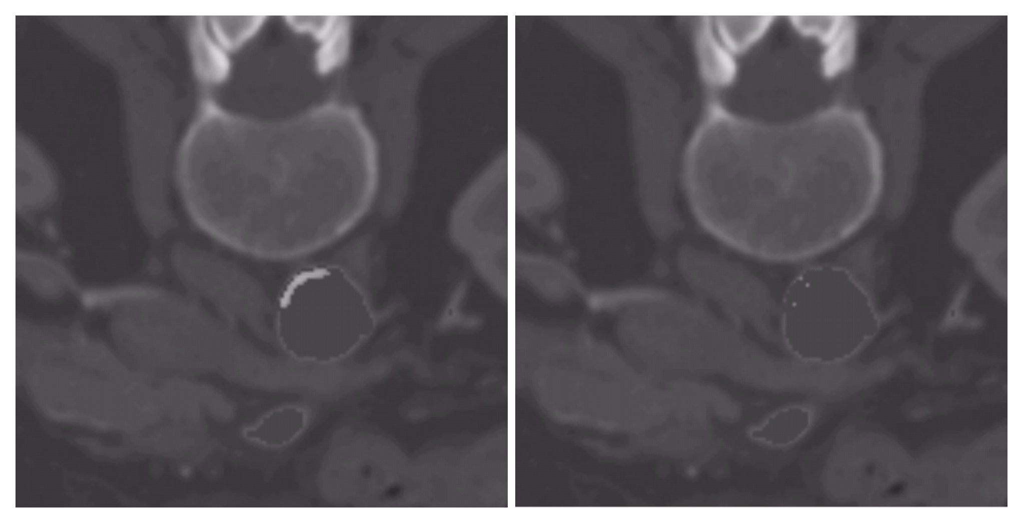


Abbildung 31: Vergleich Kalk – Schnitt Z-Achse

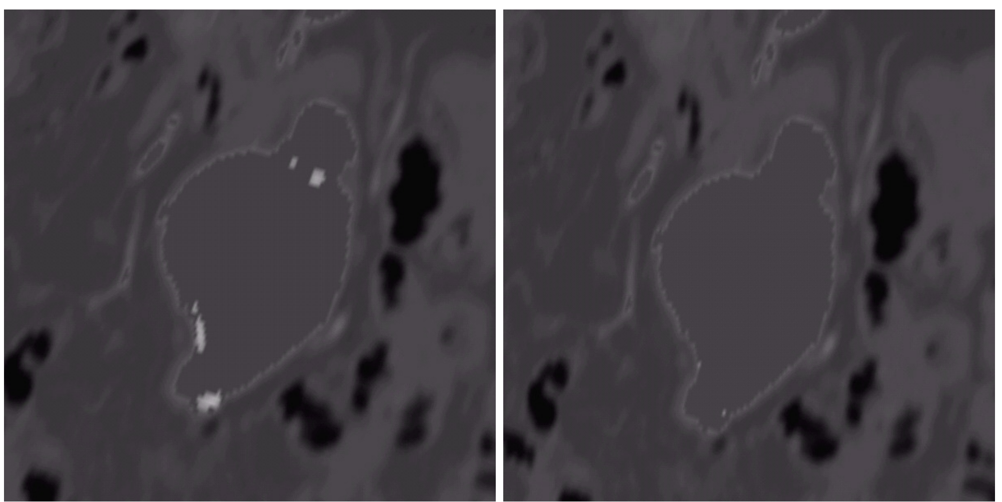


Abbildung 32: Vergleich Kalk – Schnitt Y-Achse

Eine weitere Möglichkeit die Verkalkungen zu segmentieren, wäre eine richtige Segmentierung mittels Region-Growing-Verfahren gewesen, wie es bei der Blutbahn angewandt wurde. Hierzu hätte der Benutzer allerdings einen Saatpunkt pro Verkalkung angeben müssen, damit alle Verkalkungen segmentiert werden können. Dies wäre angesichts der Vielzahl möglicher Verkalkungen zu aufwändig und fehleranfällig gewesen.

Abschließend ist zur Extrahierung der Verkalkungen zu sagen, dass das Verfahren mittels des Schwellwertfilters einfach aber wirkungsvoll ist. Es gelingt durch die auf das Umfeld der Blutbahn reduzierte Suche nahezu alle Verkalkungen zu extrahieren. Es muss allerdings berücksichtigt werden, dass der verwendete Datensatz in einer entsprechend guten Qualität bezüglich der

Schichtdicke vorliegen muss, da sonst kleinere Verkalkungen erst gar nicht auf den Schichtbildern abgebildet sind und somit auch nicht extrahiert werden können.

5.7 Visualisierung

Die Visualisierung stellt das eigentliche Ziel dieser Arbeit dar. Hierbei werden die drei segmentierten Volumen dargestellt, um Aufschlüsse über die Verkalkungen der Blutbahn zu geben. Es muss darauf geachtet werden, dass man sich bei der Betrachtung der Blutbahn und deren Verkalkungen noch orientieren kann, weshalb zusätzlich noch das Skelett dargestellt wird.

Die einzelnen Objekte werden in unterschiedlichen Farben dargestellt, um sie besser voneinander unterscheiden zu können. Die Farben symbolisieren die Art des Objekts, das dargestellt wird, so dass der Betrachter sofort erkennt, worum es sich handelt. Die Blutbahn wurde aufgrund der roten Farbe von Blut rot eingefärbt. Die Verkalkungen sind weiß, da Kalk eine weiße Farbe hat. Nur das Skelett wurde nicht seiner eigentlichen Farbe entsprechend eingefärbt, weil es nur als Unterstützung dienen soll und nicht den Betrachter von den eigentlichen zwei Objekten ablenken darf. Das Skelett wurde in einem ähnlichen blau wie der Hintergrund gehalten.

Die einzelnen Objekte werden auch mit unterschiedlichen Transparenzen dargestellt, um durch sie hindurch sehen zu können. So ist das Skelett als unwichtigstes Objekt zu neunzig Prozent transparent, um die Blutbahn und die Verkalkungen sichtbar zu machen. Die Blutbahn hingegen ist zu siebzig Prozent durchsichtig, um gegebenenfalls Verkalkungen in ihrem Inneren erblicken zu können. Die Verkalkungen an sich sind nicht transparent.

Vom Start bis zur Darstellung der Volumen vergeht eine Zeit von knapp fünf Minuten. Dabei dauert das Einlesen der drei Volumen nur knapp neunzig Sekunden. Danach folgt mit nur wenigen Sekunden die Umwandlung von ITK nach VTK. Bis zur Darstellung vergehen darauf dann nochmals etwas mehr als drei Minuten, in der die Volumen geglättet und gerendert werden. In Anbetracht der noch immer recht großen Datenmenge der Volumen handelt es sich hierbei allerdings um eine akzeptable Zeit.

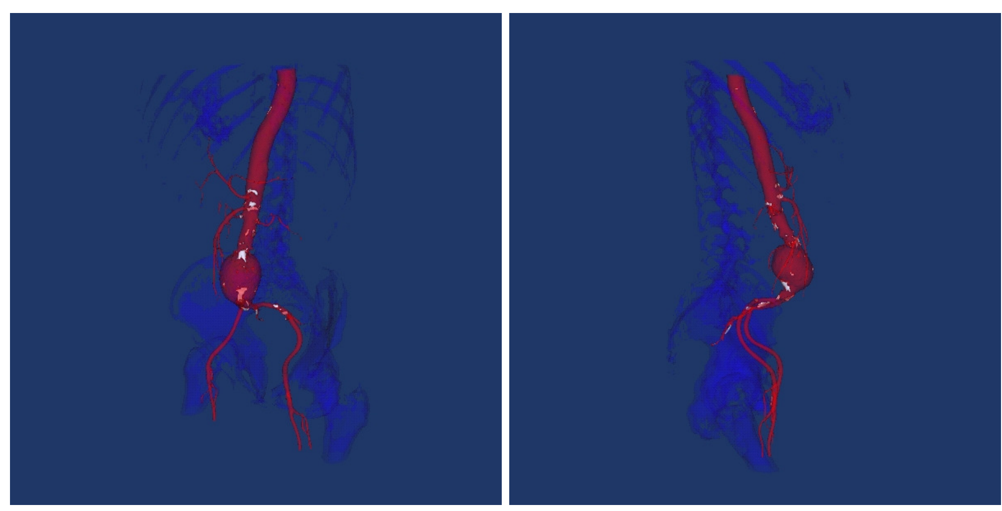


Abbildung 33: Visualisierung

Die Interaktion mit den dargestellten Volumen läuft recht flüssig. Die Darstellung stockt zwar beim Drehen oder Skalieren etwas, ist allerdings mit zwei bis drei Schritten pro Sekunde noch im Rahmen des erträglichen, um damit arbeiten zu können.

Durch die Möglichkeit, das Volumenbild frei drehen, skalieren und verschieben zu können, ist es dem Benutzer möglich, selbst kleinste Verkalkungen genau von allen Seiten zu betrachten.

6 Schlusswort

6.1 Zusammenfassung

Zusammenfassend ist letztlich zu sagen, dass eine computergestützte Diagnose von Verkalkungen der arteriellen Gefäße sehr wohl möglich ist. Voraussetzung dafür ist ein qualitativ hochwertiger Datensatz, um auch Verkalkungen geringer Größe erkennen zu können. Hat man einen guten Datensatz vorliegen, den man nicht unbedingt filtern muss, so kann man innerhalb von weniger als zwanzig Minuten zu einer Visualisierung der Blutbahn mit den dazugehörigen Verkalkungen gelangen. Dem Benutzer wird es dadurch ermöglicht, innerhalb kürzester Zeit Informationen über Engstellen in der Blutbahn zu bekommen.

6.2 Erweiterungs- und Verbesserungsmöglichkeiten

Verbesserungen sind vor allem hinsichtlich der Zeitoptimierung und Performanz zu sehen. So dauern manche Schritte, wie das Einlesen und Filtern des Datensatzes sowie das Rendern der Volumen zum Schluss noch verhältnismäßig lange.

Doch auch am Programm CADView sind hinsichtlich der Benutzerfreundlichkeit noch einige Stellen zu optimieren. So wäre es wünschenswert die Dateinamen der einzulesenden oder zu speichernden Volumen mittels eines „Datei öffnen“ oder „Datei speichern“ Dialogs angeben zu können. Dies hätte dann den Vorteil, dass bestehende Dateien einfacher gewählt werden können.

Des Weiteren wäre es noch von Vorteil, die Parameter, die beim Picker gewonnen werden, nicht per Hand einzugeben, sondern diese direkt vom Picker an die Benutzeroberfläche weiterzuleiten.

Geht man noch einen Schritt weiter, wäre es vorstellbar eine Echtzeitsegmentierung beim Picken vorzunehmen, wie es bei Sherbondy [2] umgesetzt wird. Hierbei wird direkt die angeklickte Struktur segmentiert und man kann auf Anhieb sehen, wie gut die Wahl des Saatpunktes getroffen wurde.

6.3 Danksagung

Zum Schluss möchte ich noch die Gelegenheit nutzen und mich bei Dipl. Informatiker Matthias Biedermann für seine Betreuung bedanken, der immer mit Rat und Verbesserungsvorschlägen zur Verfügung stand.

7 Anhang

7.1 Benutzerhandbuch

Zu Beginn muss das Programm CADView auf dem eigenen Rechner compiliert werden, um es danach ausführen zu können. Hierzu werden die unter Software 2.4 beschriebenen Programme benötigt. Als erstes wird empfohlen, CMake auf dem Rechner zu installieren. Damit können dann die Projektdateien der drei Toolkits erzeugt werden, um diese dann mittels Microsoft Visual .NET compilieren zu können. Ist dies geschehen, kann die Projektdatei für CADView mittels CMake erstellt werden und diese wiederum compiliert werden, wodurch eine ausführbare Datei erzeugt wird, die gestartet werden kann.

Im Folgenden wird ein Durchlauf des erstellten Programms CADView durchgegangen, um die Handhabung und Wahl der Parameter zu erklären. Es wird dabei genau gezeigt, was in welchem Schritt eingetragen werden sollte, was Standardwerte sind, die beibehalten werden sollten und wie mit der Visualisierung zur Parameterwahl interagiert werden kann.

7.1.1 Datensatz einlesen

Zum Einlesen von Datensätzen stehen wie schon erwähnt zwei Möglichkeiten zur Verfügung. Zum einen kann man DICOM-Dateien und zum anderen RAW-Dateien einlesen. Wenn der Benutzer auf den Reiter „Datensatz einlesen“ klickt, wird ihm also die Wahl geboten, welche Art von Datensatz er einlesen möchte.
Als Beispieldatensatz für das Einlesen von DICOM-Dateien wird ein Abdomenausschnitt verwendet. Der Datensatz besteht aus 288 Schnittbildern und ist 144 MB groß. Jedes Schnittbild hat eine Auflösung von 512 x 512 Pixel. Die einzelnen Schichtbilder liegen in einem Ordner vor und sind fortlaufend von IM-0001-0001.dcm bis IM-0001-0288.dcm durchnummeriert.

7.1.1.1 DICOM einlesen

Klickt der Benutzer im Hauptreiter „Datensatz einlesen“ wird ihm als erstes die Möglichkeit geboten einen DICOM-Datensatz einzulesen. Hierzu werden nur das Verzeichnis, in dem sich die Schichtbilder befinden, die Parameter für den Weichzeichner und der Dateiname, unter dem das erstellte Volumen gespeichert werden soll, benötigt.

Angenommen die Schichtbilder befinden sich in einem Verzeichnis „abdomen“ direkt auf der Festplatte „C:/“, dann lautet die Eingabe des Verzeichnisses „C:/abdomen“.

Als Parameter für den Weichzeichner werden, wie in der Auswertung beschrieben, die Standardwerte „0.125“ für die *TimeSteps*, „2“ für die Anzahl der Wiederholungen und „1“ für die *Conductance* angegeben.

Das erstellte Volumen soll unter dem Dateinamen „abdomen.vtk“ gespeichert werden.

Der Vorgang des Einlesens und Filterns der Schichtbilder ist der aufwändigste und kann unter Umständen abhängig vom Datensatz und den gewählten Parametern zwischen zehn Minuten und zwei Stunden betragen. Hierbei wird der Großteil der Rechenzeit für die Filterung des Volumens benötigt

Ist dieser Vorgang abgeschlossen, kann mit dem nächsten Schritt, dem Picker, fortgefahren werden.

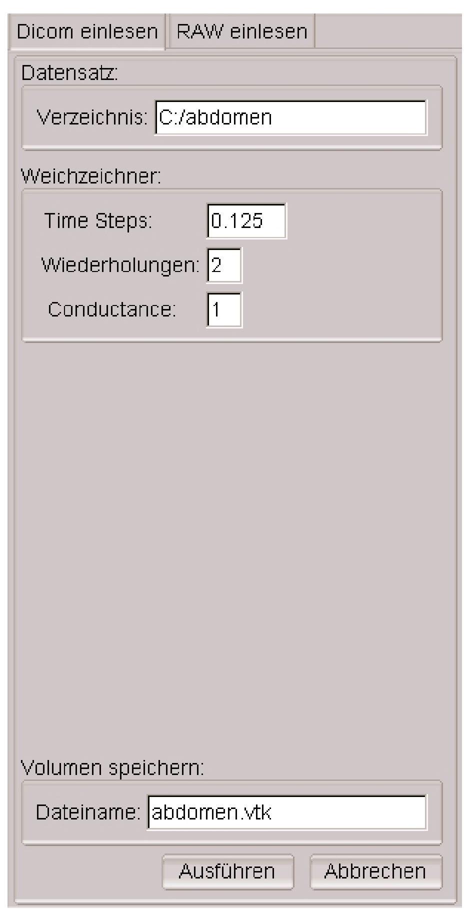


Abbildung 34: DICOM einlesen

7.1.1.2 RAW einlesen

Als Alternative zum Einlesen von DICOM-Datensätzen wird zusätzlich auch das Einlesen von RAW-Datensätzen unterstützt. Hierbei sind allerdings einige Parameter mehr als beim DICOM-Format anzugeben. So werden neben dem Dateinamen des Volumens Angaben über die Dimension des Datensatzes in jede Richtung und die Ausdehnung eines Voxels in jede Richtung benötigt.

Darauf folgen wieder wie beim Einlesen von DICOM-Datensätzen die Parameter für den Weichzeichner. Hier werden die gleichen Standardparameter für 3D-Datensätze verwendet.

Nach einem Klick auf den Button „Ausführen“ wird die Datei eingelesen, gefiltert und unter dem angegebenen Dateinamen im VTK-Format gespeichert.

Auch hier kann der Vorgang zwischen 10 Minuten und bis zu zwei Stunden oder länger betragen, abhängig von der Datensatzgröße und der Wahl der Parameter für den Filter.

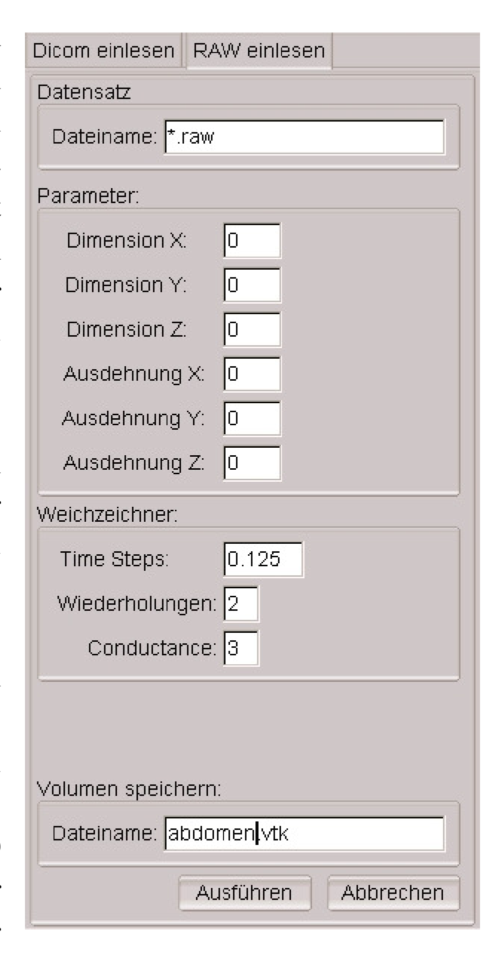


Abbildung 35: RAW-Einlesen

7.1.2 Picker

Beim nächsten Schritt, dem Picker, geht es darum Parameter für die darauf folgenden Schritte zu sammeln. Hierzu wird das zu untersuchende Volumen angezeigt und man kann Ebenen in jeder Dimension verschieben und drehen, bis man die zu untersuchende Struktur im Blick hat. Hierzu ist es im Programm unter dem Reiter „Picker“ nur notwendig den Dateinamen des Volumens anzugeben und die Darstellung mittels des „Ausführen“-Buttons zu starten. Arbeitet man die Schritte nacheinander ab, ist es nicht notwendig den Dateinamen einzutragen, da er vom Schritt zuvor übernommen wurde. Im aktuellen Beispiel lautet der Dateiname „abdomen.vtk“.
Wurde die Darstellung gestartet, erscheint nach kurzer Zeit das Bild mit den Schnittebenen durch die drei Raumachsen. Es kann mit einer Maus mit drei Tasten und der Tastatur interagiert werden.

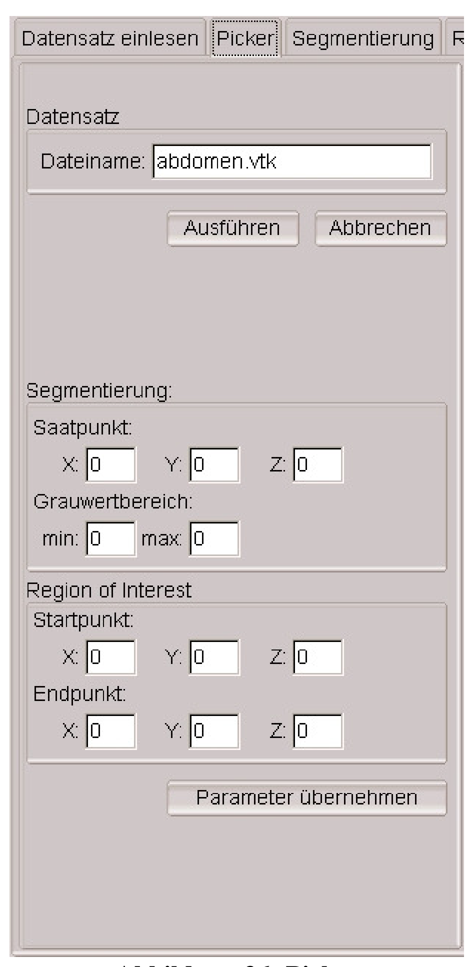


Abbildung 36: Picker

- Um das **Volumen** zu **drehen** klickt man einfach mit der linken Maustaste in den Hintergrund. Dabei dreht sich das Volumen in die Richtung, die man angeklickt hat.
- Zum **Skalieren** des Volumens klickt man mit der rechten Maustaste in den Hintergrund. Um das dargestellte Volumen zu vergrößern klickt man in den oberen Teil des Hintergrundes und um das Volumen zu verkleinern in den unteren Teil. Je nachdem wie weit man sich vom Mittelpunkt entfernt, wird stärker oder schwächer skaliert.
- Um die einzelnen **Ebenen** zu **verschieben**, klickt man mit der mittleren Maustaste auf die entsprechende Ebene. Verschiebt man nun mit gedrückt gehaltenem Rad die Maus, verschiebt sich auch die Ebene.
- Will man eine **Ebene rotieren**, so geht dies genauso, nur klickt man mit der mittleren Maustaste nicht in die Ebene, sondern an den farblich abgesetzten Rand der Ebene.
- Klickt man mit der mittleren Maustaste in den Hintergrund kann man durch Bewegen der Maus das komplette **Volumen im Fenster verschieben**

- Um die **Helligkeit und darzustellenden Graustufen anzupassen,** klickt man mit der rechten Maustaste auf das Volumen und verschiebt die Maus. Schiebt man die Maus rechts/links, verändert man die Helligkeit. Schiebt man sie auf/ab, verändert man die Graustufen. Gleichzeitig werden auch in der linken unteren Ecke die aktuellen Werte dazu angezeigt. Ein breites Spektrum, um gut die gewünschten Parameter wählen zu können, stellt die Kombination (2000,100) dar.
- Mit den Tasten „X", „Y" und „Z" kann man die entsprechende **Ebene ein- beziehungsweise und ausblenden**.
- Die Tasten „W" und „S" dienen dazu zwischen der „**Wireframe-Darstellung**" und der „**Surface-Darstellung**" umzuschalten.
- Die Taste „R" ist für einen **Reset** zuständig und skaliert wieder auf die Ausgangsgröße zurück.
- Um schließlich die **Koordinaten und Grauwerte** einer bestimmten Struktur **bestimmen** zu können, klickt man mit der linken Maustaste auf die gewünschte Struktur und bekommt in der linken unteren Ecke die Koordinaten (X, Y, Z) und den zu der Koordinate gehörenden Grauwert angezeigt.

Nachdem nun bekannt ist, wie mit dem Volumen interagiert werden kann, kann begonnen werden die benötigten Parameter zu sammeln.

Die ersten Parameter sind ein geeigneter Saatpunkt für die Segmentierung der Blutbahn und deren Grauwertbereich. Am einfachsten ist die Wahl des Saatpunktes, wenn die X- und Y-Ebenen ausgeblendet sind und man einen waagerechten Schnitt durch die Blutbahn betrachten kann. Ein geeigneter Saatpunkt ist ein Punkt mitten in der Blutbahn an einer Stelle, an der sie recht dick und gleichmäßig ist. Dies ist zum Beispiel im Bereich des unteren Brustkorbes der Fall. Im Beispielvolumen wird der Saatpunkt (282, 259, 223) mit einem Grauwert von 370 gewählt.

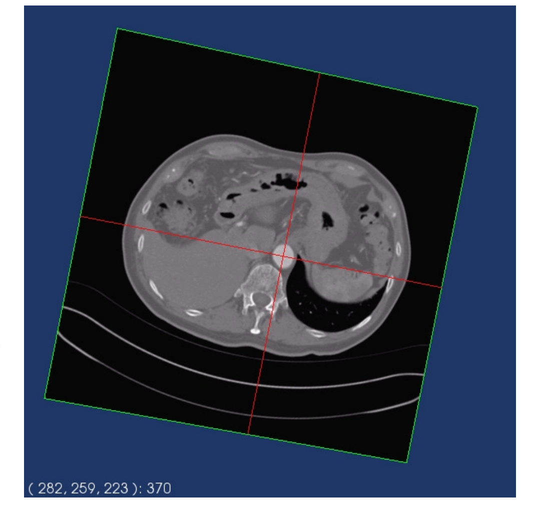


Abbildung 37: Saatpunkt wählen

Um den Grauwertbereich der Blutbahn zu erhalten, muss man die Blutbahn nach unterschiedlichen Graustufen absuchen. Vor allem die Ränder sind wichtig, um die Schwellwerte bestimmen zu können. Bei diesem Datensatz hat sich ein Grauwertbereich von ungefähr 200 bis 500 für die Blutbahn ergeben. Die so gesammelten Werte können nun in der Benutzeroberfläche in die dafür vorgesehenen Felder eingetragen werden.

Nachdem nun ein Saatpunkt gewählt wurde, kann man damit beginnen die Eckpunkte für die *Region of Interest* festzulegen. Hierbei ist darauf zu achten, das Volumen möglichst klein zu halten, aber nichts von der Blutbahn abzuschneiden. Das Vorgehen ist hierbei die angezeigten Ebenen in ihrer Minimalausdehnung und ihrer Maximalausdehnung an die Blutbahn anzupassen. Hierzu werden einmal die Ebenen so verschoben, dass sie sich in der vorderen, unteren, rechten Ecke schneiden ohne die Blutbahn zu schneiden und in der gegenüberliegenden Ecke. Die Schnittpunkte der Ebenen sind dann jeweils die Eckpunkte der *Region of Interest*. Als geeignete Eckpunkte haben sich bei diesem Datensatz die Punkte (120, 150, 0) und (400, 350, 287) herausgestellt. Diese Koordinaten können nun auch wieder in die dafür vorgesehene Stelle eingetragen werden.

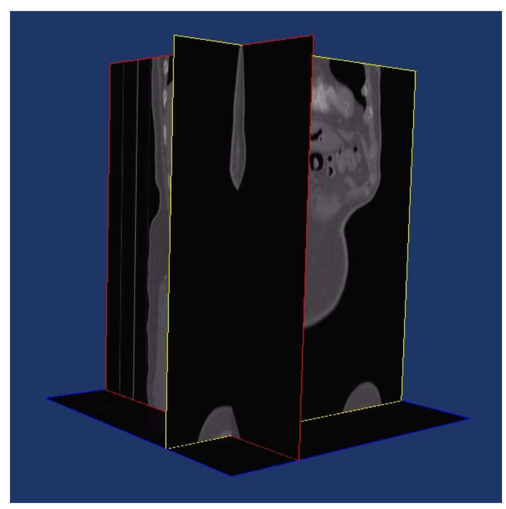


Abbildung 38: Region of Interest - Minimum

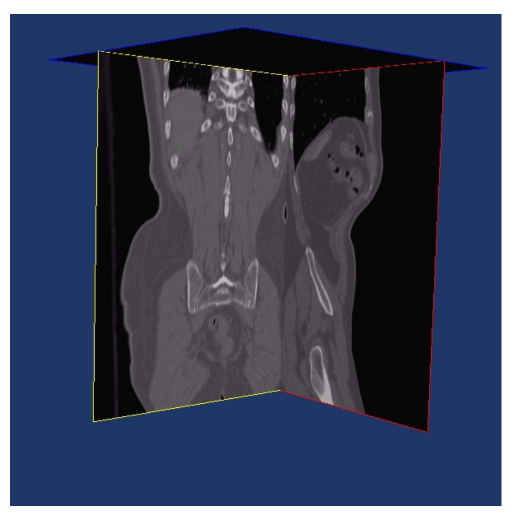


Abbildung 39: Region of Interest - Maximum

Nach Abschluss der Parameterwahl und erfolgtem Eintragen der Parameter können diese abschließend mittels des Buttons „Parameter übernehmen“ für die darauf folgenden Schritte übertragen werden.

Segmentierung:
Saatpunkt:
X: 282 Y: 259 Z: 223
Grauwertbereich:
min: 200 max: 500
Region of Interest
Startpunkt:
X: 120 Y: 150 Z: 0
Endpunkt:
X: 400 Y: 350 Z: 287
Parameter übernehmen

Abbildung 40: Parameter setzen

7.1.3 Segmentierung

Sind die Parameter übernommen worden, kann man in den Hauptreitern zum nächsten Schritt, der Segmentierung, wechseln. Hier hat man nun die Wahl zwischen zwei unterschiedlichen Segmentierungsverfahren.

7.1.3.1 Connected Threshold

Das favorisierte Verfahren ist auf dem ersten Unterreiter hinterlegt. `Connected Threshold` ist ein Schwellwertverfahren, das ausgehend von einem Saatpunkt die Region darum segmentiert, falls deren Grauwerte im angegebenen Grauwertbereich liegen.

Die benötigten Parameter für die Segmentierung der Blutbahn, wie der Dateiname des originalen Volumens, die Saatpunktkoordinaten und die Schwellwerte, wurden vom Schritt zuvor übernommen, können allerdings auch manuell gesetzt werden. Im aktuellen Beispiel wurde der Saatpunkt (282, 259, 223) mit einem Grauwertbereich von 200 bis 500 gewählt. Der *Replace Value* ist der Wert, der anstelle des originalen Grauwertes eines segmentierten Voxels gesetzt werden soll. Hier ist „255“ ein Standardwert, der für die weiteren Schritte benötigt wird und somit nicht geändert werden sollte.

Zur Segmentierung des Skelettes wird nur ein Grauwertbereich und wieder der *Replace Value* benötigt. Für den Grauwertbereich werden wieder Standardwerte benutzt, die erfahrungsgemäß den Grauwerten von Knochen entsprechen. Genauere Parameter sind nicht notwendig, da das Skelett nicht weiter von Bedeutung ist. Der *Replace Value* ist ebenfalls wieder Standard und sollte in Anbetracht der folgenden Schritte nicht geändert werden.

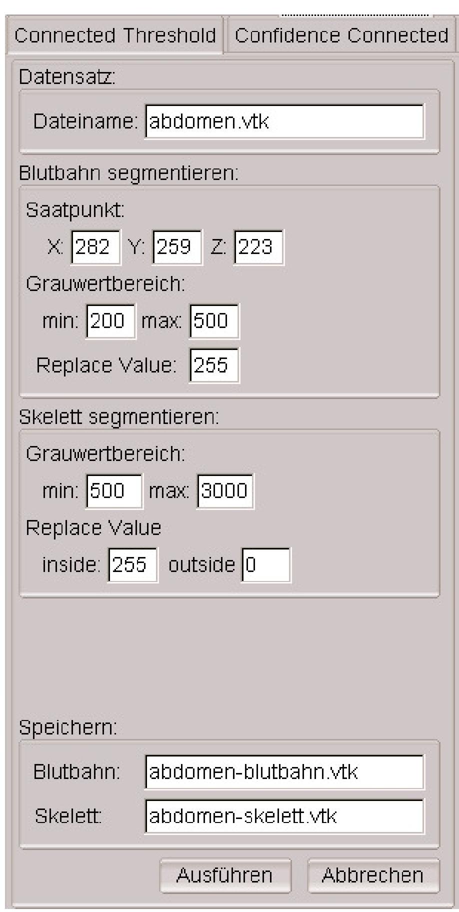


Abbildung 41: Segmentierung - Connected Threshold

Bevor die Segmentierung mittels „Ausführen“ gestartet wird, werden noch die Dateinamen benötigt, unter denen die segmentierten Volumen gespeichert werden sollen. Als Dateiname für die segmentierte Blutbahn wird „abdomen-blutbahn.vtk“ und für das Skelett „abdomen-skelett.vtk“ verwendet.

7.1.3.2 Confidence Connected

Als weitere Möglichkeit die Blutbahn zu segmentieren, wurde der `Confidence Connected` Filter implementiert. Er ist unter dem Hauptreiter Segmentierung zu finden und benötigt ein paar mehr Parameter als der Segmentierer zuvor. Neben dem Dateinamen des originalen Volumens wird hier ebenfalls ein Saatpunkt als Ausgangspunkt für die Segmentierung benötigt, der in die dafür vorgesehenen Felder einzutragen ist. Weiter benötigt er, wie in Kapitel 4 beschrieben wurde, eine Farbabweichung, mit der multipliziert wird, die Anzahl der Wiederholungen, der *Replace Value* und die Größe der Nachbarschaften. Die letzten vier Parameter sind wiederum Standardparameter, mit denen erfahrungsgemäß die besten Resultate erzielt werden können. Danach folgen dann wie beim Segmentierungsfilter davor die Parameter für die Extrahierung des Skelettes. Zum Abschluss werden wiederum die Dateinamen benötigt, unter denen die Volumen gespeichert werden sollen. Hier wird wiederum „abdomen-blutbahn.vtk“ für die segmentierte Blutbahn und „abdomen-skelett.vtk“ für das extrahierte Skelett eingegeben und die Segmentierung durch einen Klick auf den Button „Ausführen“ gestartet.

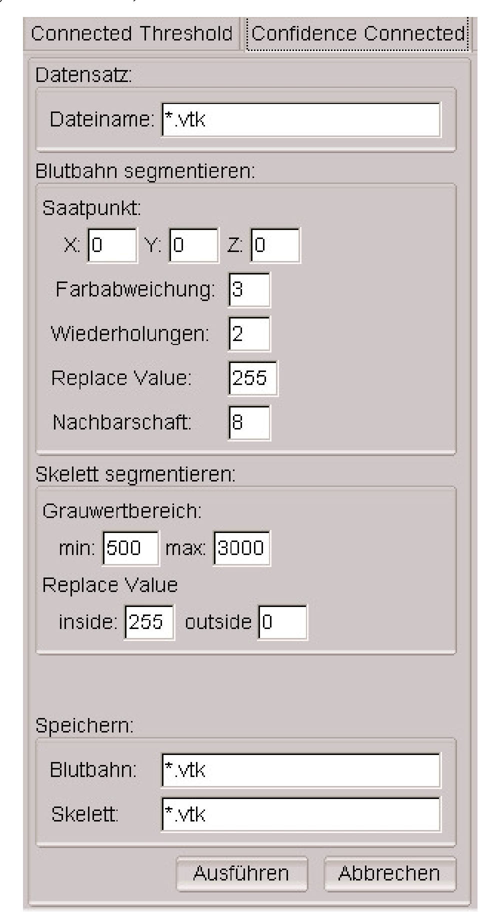


Abbildung 42: Segmentierung Confidence Connected

7.1.4 Region of Interest

Ist die Segmentierung abgeschlossen, kann mit dem nächsten Hauptschritt, der Eingrenzung des Volumens auf eine *Region of Interest,* fortgefahren werden.

Die benötigten Parameter, wie die Dateinamen der einzulesenden Volumen und die Eckpunkte der Region, wurden aus den vorherigen Schritten mit übernommen, können allerdings auch wieder manuell gesetzt werden. Es wird zum einen der Dateiname des originalen Volumens „abdomen.vtk“ sowie der Dateiname der Blutbahn „abdomen-blutbahn.vtk“ benötigt. Als Startpunkte für die Region wurde beim „Picker“ der Punkt (120, 150, 0) und als Endpunkt der Punkt (400, 350, 287) festgelegt.

Als letztes vor der Ausführung werden noch die Dateinamen der neu erstellten Volumen benötigt, unter denen diese gespeichert werden sollen. Für die *Region of Interest* des originalen Volumens wird „abdomen-roi.vtk“ und für die *Region of Interest* der Blutbahn „abdomen-blutbahn-roi.vtk“ angegeben.

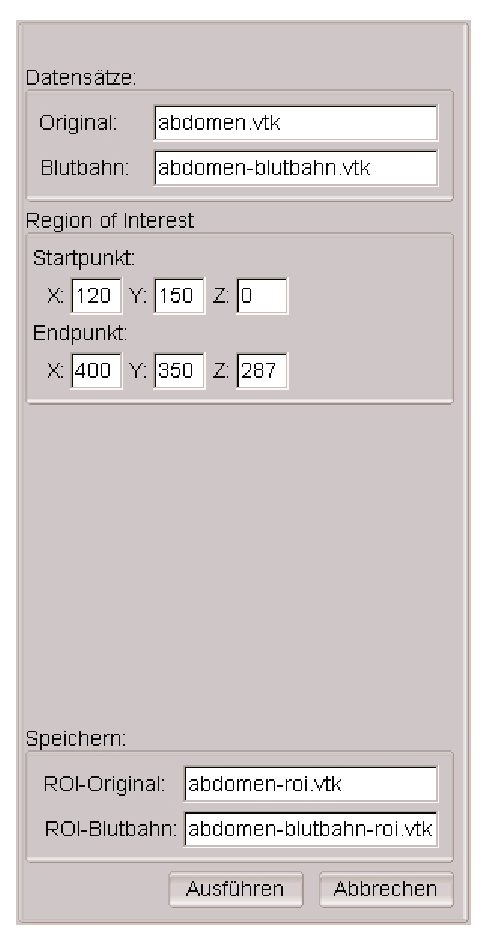


Abbildung 43: Region of Interest

7.1.5 DistanceMap

Der nächste Hauptschritt im Programm ist die Erstellung der *DistanceMap* aus der gerade erstellten *Region of Interest* der Blutbahn.
Hierzu werden nur der Dateiname der *Region of Interest* der Blutbahn und ein Dateiname, unter dem die *DistanceMap* gespeichert werden soll, benötigt. Der erste Dateiname wird von dem Schritt zuvor übernommen oder kann selbst auf „abdomen-blutbahn-roi.vtk“ gesetzt werden. Als Dateiname für die *DistanceMap* wird „abdomen-distancemap.vtk“ angegeben und die Erstellung dieser durch einen Klick auf „Ausführen“ gestartet.

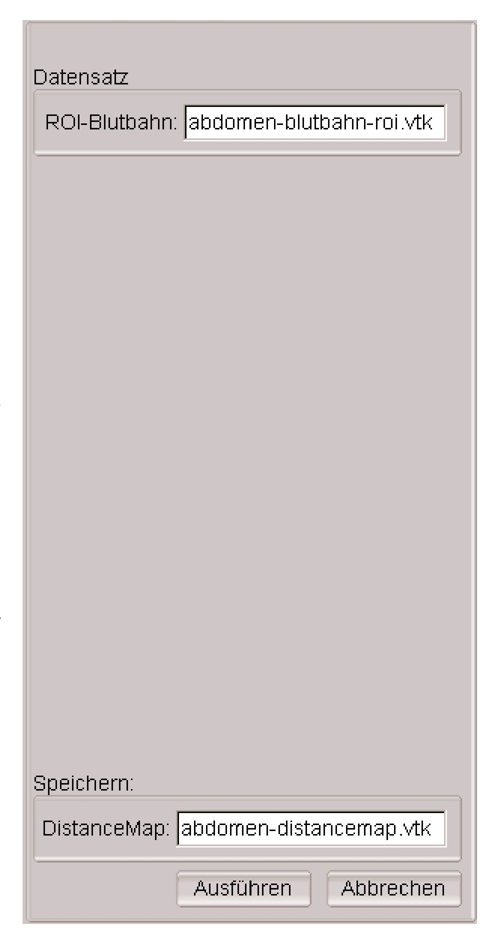


Abbildung 44: DistanceMap

7.1.6 Extrahierung

Nachdem die *DistanceMap* erstellt wurde, kann wiederum mit dem nächsten Schritt, der „Extrahierung“, fortgefahren werden. Dies ist der letzte und auch zugleich wichtigste Schritt vor der Darstellung, denn hier werden die Verkalkungen extrahiert und in einem separaten Volumen gespeichert.

Wie immer werden fast alle Parameter von den Schritten zuvor übernommen oder können manuell eingegeben werden. So werden zur Kalkextrahierung die Dateinamen der originalen *Region of Interest* „abdomen-roi.vtk“ und der *DistanceMap* „abdomen-distancemap.vtk“ benötigt. Um die Größe des Umfeldes um die Blutbahn herum, in dem nach Kalk gesucht werden soll, anzugeben, kann man einen Grauwertbereich angeben. Meist genügt ein Abstand von „1“ bis „3“ Voxel um die Blutbahn herum. Als nächstes werden noch Ersetzungswerte benötigt, die allerdings auf Standard „1“ und „0“ gesetzt bleiben sollten, um die Berechnung korrekt ausführen zu können.

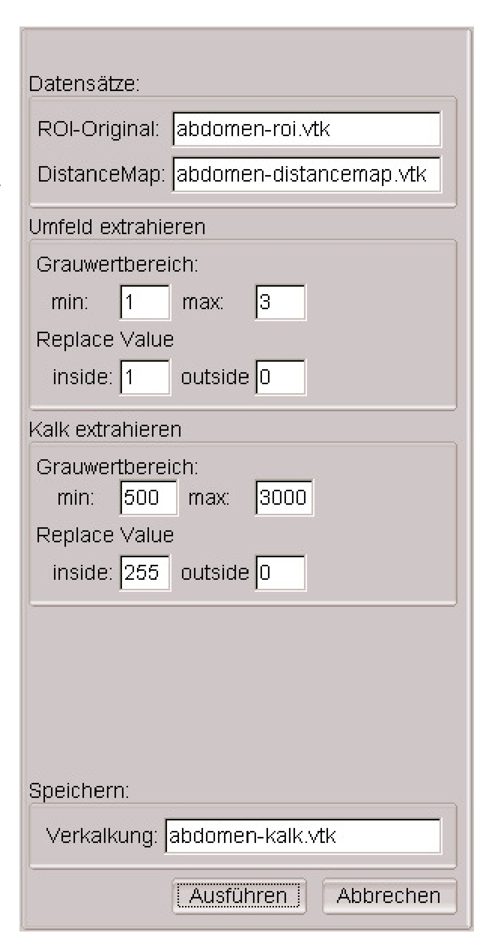


Abbildung 45: Extrahierung

Um schließlich die Verkalkungen extrahieren zu können, benötigt man den Grauwertbereich, den die Verkalkungen haben. Diese liegen zwischen den Standardwerten „500“ und „3000“. Weiter werden noch die *Replace Values* benötigt, auf welche die Verkalkungen beziehungsweise der Rest gesetzt werden sollen. Nach dem Start durch Klicken auf den Button „Ausführen“ werden dann nach kurzer Zeit die Verkalkungen unter dem angegebenen Dateinamen „abdomen-kalk.vtk“ gespeichert.

7.1.7 Visualisierung

Wenn nun alle Schritte wie beschrieben abgearbeitet wurden, liegen die drei zur Visualisierung benötigten Volumen (Blutbahn, Verkalkung und Skelett) vor.

Es müssen nur noch die Dateinamen der Volumen angegeben werden, falls diese nicht schon von den vorherigen Schritten übernommen wurden. Es werden die Volumen „abdomen-blutbahn.vtk“, „abdomen-kalk.vtk“ und „abdomen-skelett.vtk“ als Dateinamen benötigt. Zusätzlich kann man noch weitere Parameter, wie die Transparenz der einzelnen Volumen, angeben, falls man von den Standardwerten abweichende Werte benötigt. Die „Contour Value“ ist der Grauwert, der dargestellt werden soll und der standardmäßig auf „255“ festgelegt ist.

Durch den Klick auf den Button „Ausführen“ wird die Darstellung gestartet und die Volumen eingelesen und visualisiert.

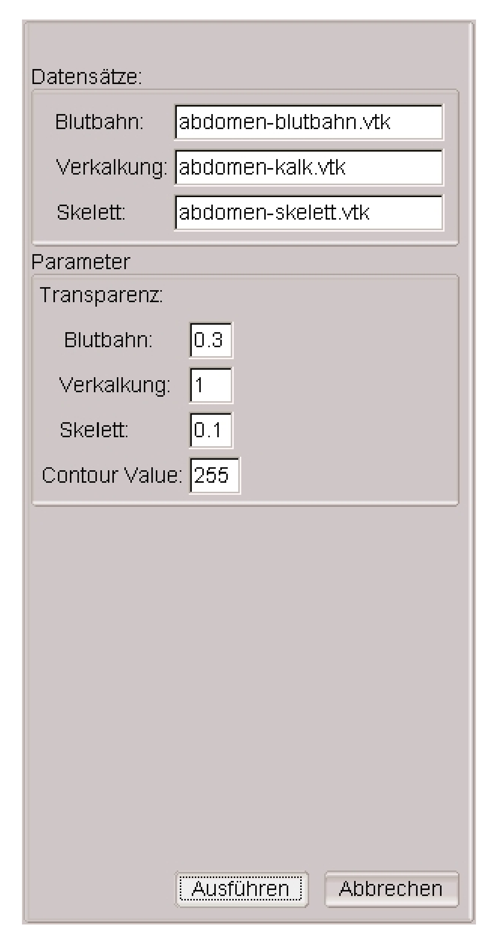


Abbildung 46: Visualisierung

7.1.7.1 Interaktion mit der Visualisierung

Mit der Visualisierung kann genauso wie im zweiten Schritt, dem „Picker“, interagiert werden:

- Das Volumen kann durch einen Klick der linken Maustaste in den Hintergrund in die entsprechende Richtung gedreht werden.
- Durch einen Rechtsklick der Maus in den Hintergrund kann das Volumen skaliert werden
- Klickt man mit der mittleren Maustaste in den Hintergrund, kann man das ganze Volumen im Fenster verschieben.

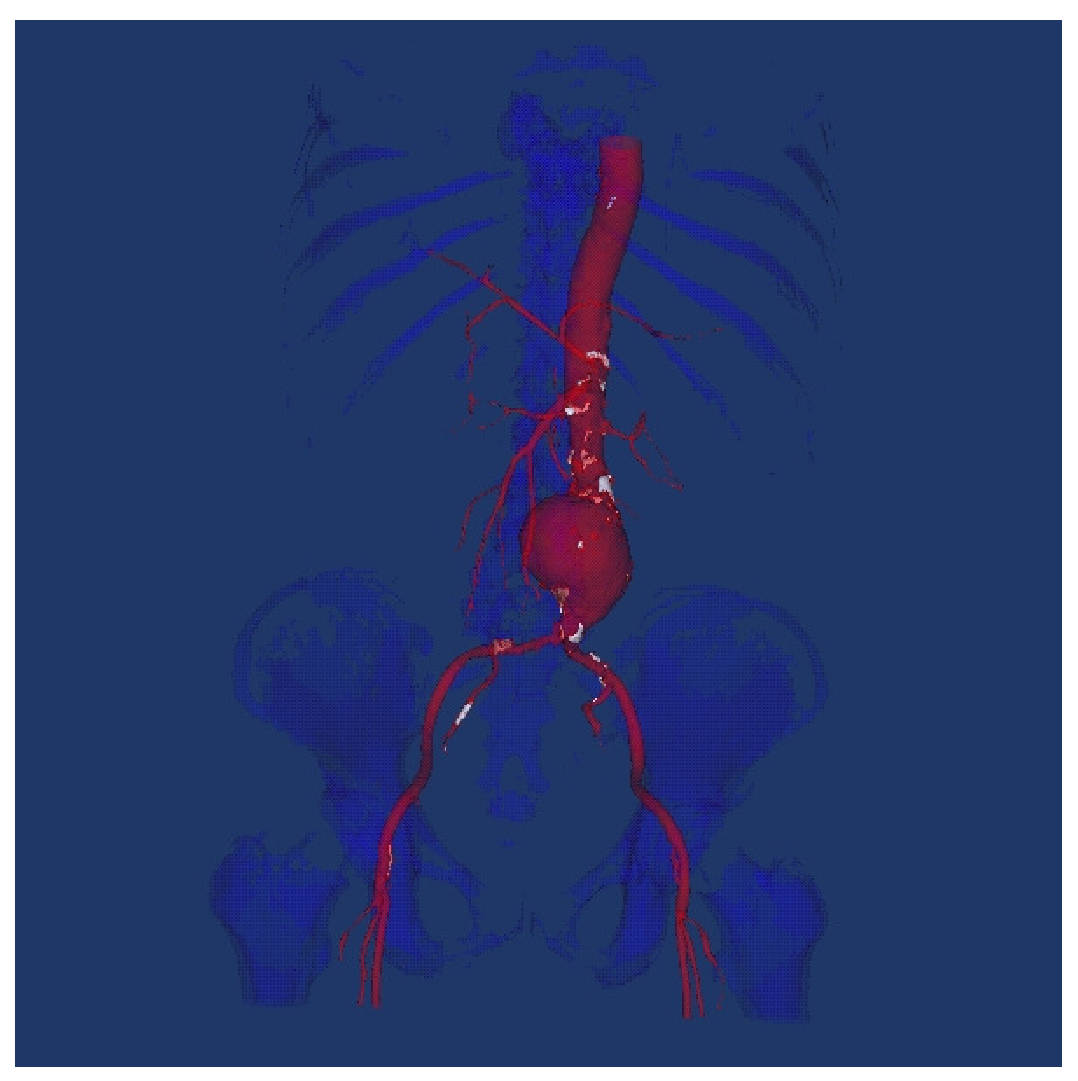


Abbildung 47: Ergebnisbild

7.2 Glossar

- **Binarisierung**

  Unter Binarisierung versteht man den Vorgang, aus einem Bild, das beliebig viele Graustufen oder andere Farbwerte hat, ein zweifarbiges Bild zu erzeugen. Hierbei werden meist ein oder mehrere Schwellwerte angegeben, welche die Grenze zwischen den Farbwertbereichen festlegen. Alle Pixel deren Farbwert unterhalb des Schwellwertes oder außerhalb des Schwellwertbereiches liegen, werden auf den einen Farbwert gesetzt, alle oberhalb des Schwellwertes oder innerhalb des Bereiches auf den anderen Farbwert.

- **DistanceMap**

  In einer *DistanceMap* wird der Euklidsche Abstand von Punkten zu dem am nächsten gelegenen Objekt angegeben. Hierbei hat dann das Objekt an sich einen Abstand von „0“ und je weiter man sich vom Objekt entfernt und somit den Abstand dazu vergrößert, wird auch der Wert am jeweiligen Pixel größer.

- **Nicht-invasiv**

  Nicht-invasiv bezeichnet in der Medizin Untersuchungsmethoden, bei denen es zu keiner Verletzung der Haut oder der Organe kommt. Typische nicht-invasive Verfahren sind beispielsweise Computertomographie oder Kernspintomographie sowie Ultraschall.

- **Picker**

  Ein Picker wird verwendet, um Objekte durch einen Sehstrahl auszuwählen. Der Sehstrahl wird dazu von der Kameraposition aus in das Grafikfenster geschossen. Die Stelle, an der der Sehstrahl die „Bounding Box“ des Actors trifft, wird als Koordinate zurückgegeben.

- **Region-Growing-Verfahren**

  Bei diesem Verfahren wird ausgehend von einem Startpunkt (Seed/Saatpunkt) eine Region segmentiert. Hierbei werden allerdings nur dem Startpunkt benachbarte Pixel untersucht und gegebenenfalls dem Volumen hinzugefügt. Die so hinzugefügten Punkte dienen auch wieder als Startpunkte und deren Nachbarn werden wiederum untersucht. Dieses Vorgehen wird so lange wiederholt, bis kein Voxel mehr dem Volumen hinzugefügt werden kann.

- **Region of Interest (ROI)**

  Eine *Region of Interest* ist ein Ausschnitt aus einem originalen Volumen, der nur den für den weiteren Verlauf besonders relevanten Bereich beinhaltet. Durch eine Eingrenzung des zu bearbeitenden Bereiches mit Hilfe solch eine Region ist es möglich, den Arbeitsaufwand der darauf folgenden Schritte erheblich zu verringern. Dadurch wird eine höhere Performanz erreicht. Zur Eingrenzung der Region werden ein Startpunkt der Region und die Ausdehnung in jede Richtung angegeben.

- **Seedpoint**

  Seedpoint ist der Ausgangspunkt für das Region-Growing-Verfahren. Dieser Punkt wird auch als Saatpunkt, Startpunkt oder einfach nur Seed bezeichnet. Abhängig von Algorithmus können auch mehrere *Seedpoints* gesetzt werden, um mehrere Regionen gleichzeitig zu segmentieren.

7.3 Abbildungsverzeichnis

7.4 Literaturverzeichnis

[1] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. IEEE Transactions on Pattern Analysis Machine Intelligence, 12:629–639, 1990.

[2] Anthony Sherbondy, Mike Houston, and Sandy Napel, "Fast Volume Segmentation with Simultaneous Visualization Using Programmable Graphics Hardware," To appear: IEEE Visualization 2003.

[3] C. Tomasi, R. Manduchi, Bilateral Filtering for Gray and Color Images, Proceedings of the1998 IEEEInternational Conference on Computer Vision, Bombay, India, 1998.

[4] Wilbur C.K. Wong, Albert C.S.Chung, Simon C.H. Yu, trilateral Filtering for Biomedical Images

[5] Will Schroeder, Ken, Martin, Bill Lorensen, “The Visualization Toolkit”, 3rd Edition, 2004. http://www.vtk.org

[6] Kitware, Inc., “VTK User’s Guide, VTK 4.4”, 2005, http://www.vtk.org.

[7] Heinz Handels, “Medizinische Bildverarbeitung”, 2000

[8] Thomas Lehmann, Walter Oberschelp, Erich Pelikan, Rudolf Repges, “Bildverarbeitung für die Medizin”, 1997

[9] Luis Ibanez, Will Schroeder, Lydia Ng, Josh Cates, The ITK Software Guide, Second Edition, http://www.itk.org, 2005

[10] FLTK Anleitungen, http://www.fltk.org

[11] CMake Anleitung, http://www.cmake.org

Erklärung zur Urheberschaft

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe. Außerdem versichere ich, dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Koblenz, 24. Oktober 2005

Alexander Horn